THÈSE DE DOCTORAT DE
l'UNIVERSITÉ PIERRE ET MARIE CURIE

Spécialité

Informatique

École doctorale Informatique, Télécommunications et Électronique (Paris)

Présentée par

Virginie LALLEMAND

Pour obtenir le grade de

DOCTEUR de l'UNIVERSITÉ PIERRE ET MARIE CURIE

Sujet de la thèse :

Cryptanalyse de chiffrements symétriques

soutenue le 5 octobre 2016

devant le jury composé de :

María Naya-Plasencia	Inria Paris	Directrice de thèse
Anne Canteaut	Inria Paris	Directrice de thèse
Henri Gilbert	ANSSI	Rapporteur
François-Xavier Standaert	UCL, Belgique	Rapporteur
Orr Dunkelman	University of Haifa, Israël	Examinateur
Pierre-Alain Fouque	Université Rennes 1	Examinateur
Antoine Joux	UPMC	Examinateur
Gregor Leander	Ruhr-Universität Bochum, Allemagne	Examinateur

Cryptanalyse de chiffrements symétriques

Virginie Lallemand

Sous la direction de María Naya-Plasencia et Anne Canteaut

Remerciements

Je ne pourrais débuter ces remerciements par quelqu'un d'autre que María Naya-Plasencia, qui sut encadrer ma thèse — et avant celle-ci mon stage — avec enthousiasme, attention et implication. Je lui suis particulièrement reconnaissante d'avoir toujours été à l'écoute et de s'être montrée patiente face à mes nombreuses questions. Je garderai un excellent souvenir de toutes les heures où nous avons travaillé ensemble et pendant lesquelles j'ai pu profiter de ses connaissances et de ses conseils mais aussi de son infatigable optimisme.

J'aimerais ensuite exprimer ma gratitude à Henri Gilbert et François-Xavier Standaert, qui ont accepté de rapporter cette thèse et de faire partie de mon jury et m'ont permis par leur relecture attentive d'améliorer le présent manuscrit.

Je tiens à remercier Anne Canteaut, Orr Dunkelman, Pierre-Alain Fouque, Antoine Joux et Gregor Leander de m'avoir fait l'honneur de participer à mon jury de thèse. Je remercie plus particulièrement Orr, Pierre-Alain et Gregor d'avoir fait le déplacement jusqu'à Paris pour assister à ma soutenance malgré leurs agendas chargés.

Je souhaiterais remercier les personnes avec qui j'ai eu l'occasion de travailler ; non seulement mes coauteurs mais aussi celles avec qui j'ai collaboré lors de séminaires ou de réunions, notamment les participants des ANR BLOC et BRUTUS. Je profite de cette occasion pour adresser ma gratitude à Itai Dinur, avec qui j'ai eu la chance de travailler lors de son passage à Paris. Thank you Itai for your patience and your kindness !

Ces années de thèse n'auraient pas été les mêmes sans la bonne humeur régnant dans l'équipe SECRET, et je souhaite donc remercier — en espérant n'omettre personne — l'ensemble des permanents, doctorants, stagiaires et amis du projet que j'ai eu la chance de côtoyer : Adrien, André, Anne, Anthony, Antoine, Audrey, Christina, Gaëtan, Ghazal, Gregory, Irene, Jean-Pierre, Joëlle (pour son amitié et nos discussions sur l'AES), Julia, Kaushik, Nicky, María, Marion, Nicolas, Pascale, Rémi, Rodolfo, Sébastien, Thomas, Valentin, Vivien, Xavier et Yann. Ma gratitude va tout particulièrement à notre cheffe de projet Anne Canteaut, dont j'admire les connaissances et la rigueur scientifique.

Merci à Christelle Guiziou, l'incroyable assistante de l'équipe, pour sa gentillesse et son efficacité pour toutes les questions administratives.

J'aimerais aussi remercier mes co-bureaux qui ont rendu mes journées de travail plus agréables encore : Gregory et Valentin (pour leur bonne humeur et leurs précieux conseils) ainsi que Sébastien, Xavier et Yann. Une mention spéciale revient à Sébastien dont les tableaux remplis de dessins de réseaux de Feistel généralisés vont sans aucun doute me manquer, quoique moins encore que nos longues discussions sur nos recherches – communes ou non.

Je souhaiterais remercier les enseignants du master Cryptis de Limoges et plus particulièrement Thierry Berger dont les cours passionnants de cryptographie symétriques (et de codes correcteurs d'erreurs) m'ont donné l'envie de poursuivre dans ce domaine.

Mon départ de l'équipe SECRET est rendu moins difficile à l'idée de rejoindre le projet UbiCrypt à Bochum pour y poursuivre mes recherches en cryptanalyse auprès de Gregor Leander et son équipe, dans un environnement de travail stimulant et enrichissant.

Comme le veut la coutume, je clôturerais ces remerciements par mes proches, donnant tout son sens à l'expression *last but not least*. Je suis infiniment reconnaissante et immensément redevable à mes parents de m'avoir toujours encouragée et soutenue dans mes projets, pour leur intérêt dans mon travail et leur affection. Merci à ma sœur d'avoir toujours été là pour moi, pour les innombrables heures passées au téléphone et pour avoir partagé l'expérience de la préparation de la thèse (elle en médecine) avec moi. Je suis incroyablement impressionnée par tout ce qu'elle réalise. Hélas, j'ai beau réécrire ce paragraphe, ces quelques mots restent bien trop faibles face à la reconnaissance que je leur porte à tous les trois.

Overview

This doctoral dissertation covers the research work in the field of symmetric cryptography that I did at Inria Paris, from 2013 to 2016, where I was a Ph.D. student in the SECRET project-team. The main topic of my research was the security analysis of symmetric primitives, with a particular focus on cryptanalysis of block and stream ciphers that have been proposed recently.

Cryptanalysis is the domain of cryptology dedicated to the security evaluation of cryptographic primitives. This domain is all the more important as for symmetric cryptography it is the only way to assess the security of primitives: namely, trust in a primitive increases if many specialists analysed the primitive without finding any flaws. Even though designers make sound security analyses before publishing their proposal, it is very important that other specialists scrutinize the cipher to give a fresh look at the construction: this is what is called *third-party analysis*.

My doctoral thesis addresses this concern by providing several analyses that in each case show that the security claim of the designers was wrong: our attacks recover the master key of the full versions of the encryption algorithms with a time complexity that is smaller than that of an exhaustive search for the secret key.

This thesis is divided in two parts: the first chapter deals with the cryptanalysis of block ciphers, and describes three cryptanalyses that are variants of the differential attack. The second studies stream ciphers and details the cryptanalysis of two recent constructions.

Part I

In the first part, we focus on recently-proposed block ciphers and analyse KLEIN, `Zorro` and PICARO. Our investigations show that they are all vulnerable to variants of differential cryptanalysis, one of the oldest — but also most powerful — statistical attacks.

Truncated differential of the block cipher KLEIN

The first cryptanalysis presented in this thesis results from a study conducted with María Naya-Plasencia on the lightweight block cipher KLEIN, a proposal presented by Zheng Gong, Svetla Nikova and Yee-Wei Law at the RFIDSec conference in 2011. The central property of our analysis, presented at FSE 2014 [LN14], is the following:

Property 3.2 *If the difference entering a round of KLEIN does not affect higher nibbles then the probability that the output difference is of the same form is* 2^{-6}.

This property gives a truncated characteristic on 1-round that is iterative and of high probability. So it can be used to build a truncated differential path that can be exploited to mount a truncated differential attack. However, the resulting truncated path on the full version of KLEIN-64 has too small probability, which prevents from using it as a distinguisher to mount a last-round attack.

Our contribution is the description of an attack that overcomes this issue by checking if the pairs of messages follow the characteristic by inverting the encryption scheme one round after the other, both in difference and in values of the lower nibbles. Thanks to additional properties of KLEIN (including properties of its key-schedule), we can do this verification in a reasonable amount of time and with only a few additional guesses. Each round inversion either keeps the number of candidates constant or reduces it. After having checked if the truncated characteristic is followed, we can also check if the computed values for the lower nibbles match the values of the messages, which further reduces the number of candidates, leading in the end to a very small set of possibilities for the value of a pair of messages that follows the characteristic, and for a part of the key. To discern the correct key we do an exhaustive search on the unknown key bits and do trial encryptions.

Our best attack on KLEIN-64 has a time complexity around 2^{10} times faster than exhaustive search. Our theoretical analysis is supported by our implementation of the attack on reduced versions of the cipher.

Cryptanalysis of substitution-permutation networks with partial non-linear layers

The second work presented here is an analysis of a construction proposed in 2013 by Benoît Gérard, Vincent Grosso, María Naya-Plasencia and François-Xavier Standaert. Their aim was to build a cipher based on the AES that is easy to protect against side-channel attacks, more precisely when the counter-measure used is the masking scheme proposed in 2010 by Rivain and Prouff. This constraint imposes to limit the number of non-linear operations, which in turn results both in finding good cryptographic S-boxes that can be expressed with a minimum number of non-linear operations and in limiting the number of S-box applications.

Their researches end up with the definition of substitution-permutation network ciphers for which only a small part of the state is modified by S-boxes. Their concrete proposal is a cipher named `Zorro`: its main difference from the AES is the fact that only 4 S-boxes are applied at every round; only in the first line of the state. The designers of `Zorro` gave a comprehensive security analysis of their design, including an analysis of its resistance against linear and differential cryptanalysis. However, given the particularity of their cipher, they were not able to use common analysis tools (like wide-trail-strategy arguments) but they used a technique based on the notion of degree of freedom. As showed by a paper published shortly after, this analysis was wrong and the authors missed iterative differential characteristics that can be exploited in an attack of the full version of the cipher.

The researches I did with Achiya Bar-On, Itai Dinur, Orr Dunkelman, Nathan Keller and Boaz Tsaban aimed at better understanding this issue. They have led to the following results, described in an article that have been presented at Eurocrypt 2015 [BDD+15]:

— We developed a generic tool dedicated to the analysis of SPN with partial non-linear layers against basic linear and differential attacks; namely, our tool can determine if there exist characteristics with a active S-boxes on r rounds.

— By applying this tool to Zorro, we were able to find its best characteristics and subsequently to mount a practical [1] (and experimentally-verified) differential attack. It uses a key-recovery step that is made to take advantage of the low non-linearity of Zorro.
— Our work pointed out that the weakness of Zorro against linear and differential attacks originates from the interaction between the *MixColumns* and the *ShiftRows* operations, both of order 4, that leads to the existence of iterative characteristics on 4 rounds.
— By deviating from the AES design strategy and by choosing a different linear operation, we have been able to build a variant of Zorro that resists basic differential and linear attacks.

This last point is essential: it shows that the general construction proposed by Gérard et al. is not inherently flawed but can be implemented in a way such that the cipher has properties that are close to what can be expected from an ideal construction.

Related-key attacks on PICARO

The last block cipher we analysed is PICARO, a 12-round Feistel construction addressing the need for ciphers that are easy to mask (note that PICARO was designed one year earlier than Zorro). The designers - Gilles Piret, Thomas Roche and Claude Carlet - mainly focused on the design of the S-Box, and finally chose an S-Box that is non-bijective. Anne Canteaut, María Naya-Plasencia and I showed in [CLN15] that the properties of the S-Box, combined with a simple key-schedule algorithm with relatively low diffusion, can lead to undesirable properties in a related-key scenario. Namely, we have exhibited a set of master-key differences W for which we have the following property:

Property 5.2 *The probability that two related keys with a difference in the set W encrypt two messages into the same ciphertext is* 2^{-126}.

We then extended this distinguisher into a key-recovery attack on the full version of the cipher. This cryptanalysis breaks the claim of the authors, which stated that their construction resists related-key attacks.

Part II

The second part of this thesis gives a security analysis of two stream ciphers proposed in 2015: the first one, Sprout, is a lightweight cipher presented at FSE 2015. The second one, FLIP, is a family of stream ciphers which first sets of parameters have been presented at a national workshop. It is commonly admitted that the design of secure stream ciphers is less understood than the design of block ciphers, but as showed by their respective designers, opting for stream ciphers was the most appropriate choice given the constraints of the targeted applications. Both designs move away from existing ciphers, and if our analyses do not question their global approach they show that some of the proposed sets of parameters do not provide the level of security claimed by the designers.

1. The attack has a time complexity close to 2^{45} and requires less than 2^{42} chosen plaintexts

Cryptanalysis of Sprout

The stream cipher Sprout was proposed by Frederick Armknecht and Vasily Mikhalev with the aim of being a lightweight primitive that both enjoys a high throughput and an efficient hardware implementation in terms of area. Their research resulted in the proposition of a new type of stream ciphers for which the key intervenes not only during the initialization process but also during the keystream generation phase. Their solution yields a stream cipher with a reduced area in comparison with usual designs while still offering the same level of security. To prove the performance gain of their approach, they describe and implement a concrete cipher that follows their new design directions. They named it Sprout, as a reference to the Grain cipher on which it is based.

The analysis that María Naya-Plasencia and I have conducted, published at Crypto 2015 [LN15] shows that the security of their cipher is lower than what they claimed.

The three properties of Sprout that we use in our attack are the following:

1. The two registers that the internal state is made of are almost independent.
2. These two registers have a small size in comparison with the key.
3. During the keystream generation, the impact of the key in the update of the registers is non-linear.

By combining these properties with a procedure approaching divide-and-conquer techniques, we reduce the search for the value of the internal state to a problem of merging two lists under some given conditions. Our attack has very low data complexity and a time complexity which is roughly 2^{10} times smaller than the cost of an exhaustive search for the key.

Cryptanalysis of the early version of the FLIP family of stream ciphers

The last work presented here is a cryptanalysis of the FLIP family of stream ciphers, a new construction proposed by Pierrick Méaux, Anthony Journault, François-Xavier Standaert and Claude Carlet. The idea that guided their research was to design a cipher that can be integrated into a hybrid fully homomorphic encryption scheme, which implies to look for a construction that interoperates well with the asymmetric part of the scheme, meaning in this case that it does not increase too much the level of noise.

The FLIP family of stream ciphers uses a new structure named *filter permutator*, which works in the following way: at each clock, the permutation generator — initialised with a public IV and associated with a PRNG — produces a public permutation that shuffles the key bits. The rearranged key bits enter a Boolean function (F) that computes the keystream bit.

To best meet the application requirements, the designers opted for a key that is only permuted and for a filtering function F that has few monomials only and a low degree. As showed by the analysis I have conducted with Sébatien Duval and Yann Rotella, published at Crypto 2016 [DLR16] the set of parameters initially proposed does not provide enough security. Namely, we were able to mount a guess-and-determine attack that takes advantage of both following remarks:

— Since the key is only permuted, a guess of one bit at a given time gives a 1-bit information at any other time.

— There are very few monomials of degree greater than or equal to 3 in F.

Our attack then proceeds as follows:

1. We start by guessing enough null key-bit positions to ensure that with high probability the permutation will position at least one null key bit into each monomial of degree greater than or equal to 3.
2. We wait for such permutations and collect quadratic equations.
3. We solve the system.

If our initial guess was correct and that the picked key bits were indeed null, this process returns the right key. Otherwise, the resolution of the system will fail.

This attack has a time complexity roughly equal to the square root of the key size. Thanks to our comments, the designers were able to come up with a new instantiation of the filter permutator that resists this type of attacks. The improved design was presented by Pierrick Méaux, Anthony Journault, François-Xavier Standaert and Claude Carlet at Eurocrypt 2016.

Introduction générale

Les travaux de recherche présentés dans cette thèse se situent en cryptographie symétrique. Plus précisément, ils apportent leur contribution à l'un des sous-domaines majeurs de la cryptographie symétrique, à savoir la cryptanalyse.

La première partie de cette thèse est dédiée à l'analyse de propositions récentes de chiffrements par blocs. Les attaques que nous proposons ici sont toutes des variantes de l'attaque différentielle : nous proposons une cryptanalyse différentielle tronquée sur le chiffrement à bas coût KLEIN, une attaque différentielle « simple » sur le chiffrement `Zorro` ainsi qu'une attaque différentielle à clefs liées sur le chiffrement PICARO.

Notre premier résultat est la cryptanalyse de la famille de chiffrements KLEIN, un ensemble de trois primitives à bas coût introduit en 2011 avec l'objectif d'atteindre de bonnes performances logicielles. L'analyse de cette construction, menée avec María Naya-Plasencia et publiée à FSE 2014 [LN14], tire parti de plusieurs propriétés de KLEIN mises en avant dans de précédents travaux, dont notamment une cryptanalyse différentielle tronquée datant de 2011. Une des principales faiblesses de KLEIN émane de l'interaction de l'opération *MixColumns* (orientée octets et provenant de l'AES) avec le reste de la fonction de tour — orientée quartets. Le problème ici provient de la faible diffusion apportée par l'opération *MixColumns* qui résulte en l'existence d'une caractéristique différentielle tronquée itérative sur 1 tour. Notre analyse montre comment cette caractéristique peut être étendue en une caractéristique différentielle tronquée de probabilité raisonnable sur l'ensemble des tours privés du dernier, puis être développée en une attaque de type différentielle tronquée. Un des apports majeurs de nos travaux est la description d'une technique permettant de retrouver la clef bien qu'il soit très difficile de distinguer les paires ne suivant pas la caractéristique. Notre technique est la première permettant d'attaquer la version complète de KLEIN avec une clef de 64 bits, et permet d'attaquer plus de tours que les précédentes attaques sur les versions utilisant des clefs plus grandes.

Le second résultat de cette thèse, décrit au chapitre 4, porte sur le chiffrement `Zorro`. Ce chiffrement fut proposé par Benoît Gérard, Vincent Grosso, María Naya-Plasencia et François-Xavier Standaert et a comme spécificité de modifier de façon non-linéaire seulement 4 octets (sur les 16 que compte l'état). Ce choix de conception — justifié par l'objectif de proposer un chiffrement facile à masquer — avait impliqué que les concepteurs n'avaient pas pu utiliser les algorithmes d'analyse usuels pour évaluer leur construction. Avec mes coauteurs Achiya Bar-On, Itai Dinur, Orr Dunkelman, Nathan Keller et Boaz Tsaban, nous avons fourni le premier outil d'analyse de ce type de construction face aux attaques différentielles et linéaires de base. En appliquant notre outil à `Zorro` nous avons pu trouver ses meilleures caractéristiques

différentielles que nous avons ensuite adaptées en une attaque pratique sur le chiffrement. Cet outil s'est aussi avéré précieux pour répondre à une question ouverte, à savoir déterminer s'il existe une variante de `Zorro` qui soit résistante aux attaques différentielles et linéaires de base. Nous avons répondu par l'affirmative à cette question en mettant en évidance un tel chiffrement. De plus, notre analyse a permis de déterminer quel était le problème structurel de `Zorro`, et nous avons pu montrer que sa faiblesse face aux attaques différentielles et linéaires (et notamment l'explication de la présence de caractéristiques itératives sur 4 tours) provenait de l'interaction entre les opérations *MixColumns* et *ShiftRows*, toutes deux d'ordre 4. Nos résultats furent publiés à Eurocrypt 2015 [BDD+15].

Le dernier chapitre de cette partie présente une attaque à clefs liées sur la construction de Feistel PICARO, proposée en 2012 (soit 1 an avant `Zorro`) par Gilles Piret, Thomas Roche et Claude Carlet. L'objectif visé par cette construction était là aussi d'être facile à protéger contre les attaques par canaux auxiliaires, toujours en considérant un masquage booléen. Cette contrainte fit opter les auteurs pour une boîte-S non bijective, pouvant être exprimée avec peu de multiplications non-linéaires, et leur imposa de construire leur chiffrement par blocs en suivant un réseau de Feistel. La cryptanalyse développée avec Anne Canteaut et María Naya-Plasencia [CLN15] repose essentiellement sur deux caractéristiques du chiffrement à savoir la non-bijectivité de la boîte-S et la faible diffusion de l'algorithme de cadencement de clef. Notre point de départ était l'observation que ces propriétés impliquent qu'une différence dans la clef-maître créera peu de différences dans les sous-clefs. Ces différences pourront donc être annulées par passage dans les boîte-S non bijectives. En exploitant cette idée, nous parvenons à créer un distingueur sur la version complète du chiffrement, puis à monter une attaque à clefs liées, là encore sur la version complète de l'algorithme.

Les dernières recherches que j'ai menées durant ces 3 ans sur les chiffrements par blocs portent sur la formalisation des attaques différentielles impossibles et font suite à l'article de Boura et al. [BNS14]. Une des questions traitées par nos recherches est la prise en compte de la définition du cadencement de clef dans le calcul de la complexité d'une attaque. Les nombreux résultats de ces recherches, menées avec Christina Boura, María Naya-Plasencia et Valentin Suder, ont été consignés dans un article en cours de soumission que nous ne présenterons pas ici.

La seconde partie du manuscrit porte sur la cryptanalyse de chiffrements à flot et étudie deux propositions récentes : Sprout et FLIP. Toutes deux sont des instanciations de nouvelles constructions générales proposées pour répondre à des applications spécifiques : Sprout est un chiffrement à bas coût conçu de façon à limiter la taille du circuit matériel nécessaire à son implémentation, tandis que l'objectif de FLIP est d'être adapté à une utilisation dans un schéma de FHE hybride.

L'idée développée par Frederik Armknecht et Vasily Mikhalev pour permettre une taille de circuit plus petite consiste à utiliser une partie de l'état interne pour stocker la clef. Cette dernière intervient ensuite dans la mise à jour des autres registres lors de l'étape de génération de la suite chiffrante. L'instance concrète de ce nouveau schéma, nommée Sprout, a comme particularité le fait que la clef intervient de façon non-linéaire. Comme nous le montrons au chapitre 7, cette spécificité combinée à d'autres caractéristiques de Sprout (comme sa structure utilisant deux registres presque indépendants) permet de monter une attaque de type diviser pour mieux régner ramenant la détermination de la valeur de l'état interne à un

problème de fusion de listes. Cette technique, développée avec María Naya-Plasencia, permet de retrouver la valeur de l'état interne et de la clef environ 2^{10} fois plus rapidement qu'une recherche exhaustive de la clef. Elle fut présentée à la conférence Crypto 2015 [LN15].

Le dernier résultat présenté dans ce manuscrit est une attaque de la première version de paramètres proposée en instanciation concrète du schéma dit *filter permutator* introduit par Pierrick Méaux et ses coauteurs. Les applications visées ici par les concepteurs sont les schémas de FHE hybride, ce qui se traduit par des contraintes sur la profondeur multiplicative du chiffrement. Dans l'article [DLR16] écrit avec Sébastien Duval et Yann Rotella, nous montons une attaque de type guess-and-determine dont le principe est de réaliser une série d'hypothèses pour réduire le degré de l'expression de certains bits de la suite chiffrante. Après avoir collecté suffisamment d'équations nous résolvons le système par des techniques de linéarisation et retrouvons la clef. Grâce à notre analyse, les concepteurs de FLIP purent améliorer leur proposition et proposer une nouvelle instanciation concrète plus résistante.

Articles publiés durant cette thèse

[DLR16] Sébastien Duval, Virginie Lallemand and Yann Rotella.
Cryptanalysis of the FLIP Family of Stream Ciphers.
In *Advances in Cryptology - CRYPTO 2016*, volume 9814 of LNCS, pages 457–475, Springer, 2016.

[CLN15] Anne Canteaut, Virginie Lallemand, and María Naya-Plasencia.
Related-Key Attack on Full-Round PICARO.
In *Selected Areas in Cryptography - SAC 2015*, volume 9566 of LNCS, pages 86–101, Springer, 2016.

[LN15] Virginie Lallemand and María Naya-Plasencia.
Cryptanalysis of Full Sprout.
In *Advances in Cryptology - CRYPTO 2015*, volume 9215 of LNCS, pages 663–682, Springer, 2015.

[BDD+15] Achiya Bar-On, Itai Dinur, Orr Dunkelman, Virginie Lallemand, Nathan Keller, and Boaz Tsaban.
Cryptanalysis of SP Networks with Partial Non-Linear Layers.
In *Advances in Cryptology - EUROCRYPT 2015*, volume 9056 of LNCS, pages 315–342, Springer, 2015.

[LN14] Virginie Lallemand and María Naya-Plasencia.
Cryptanalysis of KLEIN.
In *Fast Software Encryption - FSE 2014*, volume 8540 of LNCS, pages 451–470, Springer, 2014.

Table des matières

Chapitre 1

Introduction

La cryptologie est une discipline très ancienne s'attachant à assurer la sécurité des communications. Les premières méthodes de chiffrement connues remontent à l'Antiquité et étaient artisanales. Les techniques de protection des messages se sont rapidement améliorées pour devenir mécaniques, puis informatiques. D'abord réservée à un usage militaire et diplomatique, la cryptologie se développa jusqu'à devenir une science et un domaine de recherche académique très dynamique. Cette évolution remonte au milieu des années 1970 avec notamment la conception du *Data Encryption Standard* (DES) et l'invention de la cryptographie à clef publique par Whitfield Diffie et Martin Hellman [DH76].

1.1 Principes de base de la cryptographie

La cryptographie permet d'apporter des réponses à une situation dans laquelle deux protagonistes — que l'on nomme habituellement Alice et Bob — souhaitent communiquer de façon sûre en utilisant un canal qui ne l'est pas. Elle permet en effet de s'assurer de la confidentialité [1], de l'authenticité [2] ainsi que de l'intégrité [3] des messages échangés.

La confidentialité d'une communication est assurée par le chiffrement, procédé permettant de transformer un message d'origine (le *clair*) en un message illisible (le *chiffré*) pour toute personne ne possédant pas le secret (la *clef*) permettant d'inverser le processus pour le rendre lisible à nouveau (de le *déchiffrer*).

On distingue deux types d'algorithmes de chiffrement : ceux à clef publique (ou asymétriques) et ceux à clef privée (ou symétriques). Dans le premier cas, les deux protagonistes possèdent chacun une paire de clefs comprenant une clef privée (qui ne doit rester connue que de son propriétaire) et une clef publique (accessible par tous). Si Bob souhaite envoyer un message à Alice, il le chiffre en utilisant la clef publique d'Alice. À réception, celle-ci déchiffre le message en utilisant sa clef privée. Ce procédé est souvent introduit par la métaphore suivante : pour chiffrer son message pour Alice, Bob le met dans une boîte qu'il ferme avec un cadenas fourni par Alice (c'est la clef publique de celle-ci). Seule Alice possède la clef de ce cadenas (sa clef privée) et pourra donc ouvrir la boîte pour avoir accès au message.

Les chiffrements symétriques n'utilisent, eux, qu'une clef. Elle est partagée par les deux protagonistes et leur permet à la fois de chiffrer et de déchiffrer. L'inconvénient majeur de

1. Uniquement les personnes en ayant le droit ont accès à l'information.
2. L'identité des correspondants a été vérifiée.
3. Le message n'a pas été modifié.

cette technique est qu'elle impose de réaliser au préalable un échange sécurisé des clefs . Elle possède cependant un avantage considérable par rapport à la cryptographie asymétrique, notamment quant au temps de traitement et à la consommation énergétique. Ces paramètres font que, dans la plupart des cas pratiques, on combinera les deux types de cryptographie : on utilisera un système asymétrique pour échanger la clef secrète puis on passera à un système symétrique pour le reste de la communication. C'est ce qu'on appelle la cryptographie hybride.

Dans ce manuscrit nous nous intéressons aux systèmes de chiffrement symétriques. Ils se répartissent en deux catégories : les chiffrements à flot et les chiffrements par blocs.

1.2 Chiffrements à flot et chiffrements par blocs

Chiffrements à flot

L'idée des chiffrements à flot est d'imiter le chiffrement de Vernam (appelé aussi masque jetable), une technique permettant d'apporter une sécurité théorique absolue mais délicate à mettre en œuvre. Ce chiffrement parfait consiste à combiner par l'opération booléenne `XOR` (OU EXCLUSIF) les bits du message clair avec une suite binaire (appelée aussi suite chiffrante et qui, dans ce cas-ci, correspond à la clef). Elle doit être aussi longue que le message, totalement aléatoire et jamais réutilisée : comme chaque bit de clair a été additionné à un bit valant avec même probabilité 0 ou 1, déchiffrer sans la clef devient impossible et même une recherche exhaustive de la clef est mise en échec. Cependant, le chiffrement de Vernam est impraticable : non seulement il demande de créer une suite binaire totalement aléatoire aussi longue que le message mais il nécessite aussi de transmettre celle-ci entre les deux parties communicantes.

Les chiffrements à flot permettent de résoudre le problème de la taille et de la transmission de la clef de la façon suivante : on rend public un générateur pseudo-aléatoire, c'est-à-dire un algorithme permettant de produire une suite binaire *proche* de l'aléa[4] puis on confie à Alice et à Bob une (même) clef secrète, typiquement d'une centaine de bits. Lorsque Alice voudra envoyer un message à Bob, elle utilisera cette clef ainsi qu'une petite séquence binaire publique (appelée IV pour *Initialisation Vector*) pour paramétrer son générateur pseudo-aléatoire et obtenir une suite chiffrante. Elle combinera ensuite cette dernière à son message clair pour obtenir le chiffré qu'elle transmettra à Bob. Pour déchiffrer, Bob initialisera son générateur pseudo-aléatoire avec la même clef secrète et le même IV pour obtenir la même suite chiffrante qu'Alice. Il lui suffira alors de combiner celle-ci au chiffré pour retrouver le clair.

Si cette méthode permet de résoudre le problème de la taille et de la transmission de la clef, elle apporte néanmoins une sécurité réduite : la suite n'est plus parfaitement aléatoire et le chiffrement a une sécurité au plus définie par la taille de la clef. Nous verrons une définition des chiffrements à flot plus détaillée au chapitre 6.

Chiffrements par blocs

L'autre grande catégorie de chiffrements symétriques repose sur les chiffrements par blocs. Ils prennent en entrée un bloc de message de taille n bits (le plus souvent comprise entre 64 et 128 bits) ainsi qu'une clef K de k bits et retournent un chiffré de même taille que le clair.

4. Dans le sens où elle présente certaines propriétés des suites purement aléatoires.

Comme nous le verrons au chapitre 2, un chiffrement par blocs définit un ensemble de 2^k permutations, et choisir une clef revient à choisir la permutation utilisée.

Pour utiliser un chiffrement par blocs il sera nécessaire de définir un *mode opératoire* précisant l'enchaînement des traitements appliqués aux différents blocs du message clair.

Nous donnerons un aperçu des techniques de construction des chiffrements par blocs au chapitre 2.

1.3 Cryptanalyse

La cryptanalyse est le sous-domaine de la cryptologie s'appliquant à évaluer la sécurité des primitives cryptographiques. Son rôle est d'étudier au plus près les primitives pour s'assurer que les nouveaux schémas apportent bien le niveau de sécurité annoncé par les concepteurs.

En cryptographie symétrique, et lorsque la primitive étudiée est un chiffrement, son objectif sera le plus souvent de retrouver la clef grâce à une procédure moins coûteuse que sa recherche exhaustive. D'autres résultats comme la mise en avant d'un biais statistique dans la suite chiffrante produite par un chiffrement à flot seront aussi considérés comme des cryptanalyses, dans la mesure où ils mettent en évidence des propriétés non désirées d'un chiffrement.

Le rôle de la cryptanalyse est absolument primordial en cryptographie symétrique puisqu'elle constitue le seul moyen d'évaluation de la sécurité des chiffrements. Concrètement, plus un chiffrement aura été analysé [5] sans montrer de faiblesses et plus la communauté cryptographique aura confiance en ce chiffrement.

Au-delà de son rôle d'évaluation de la sécurité des systèmes, la cryptanalyse et ses avancées permettent d'aboutir à de nouveaux principes et critères pour construire des systèmes plus résistants.

1.3.1 Modèles d'attaques

On peut définir plusieurs types d'attaques selon l'information à disposition de l'attaquant et ses capacités.

On distingue notamment (de l'hypothèse la plus faible à la plus forte) :

— Les *attaques à chiffrés seuls*, pour lesquelles seuls des chiffrés sont à disposition de l'attaquant. Ce modèle est le plus difficile à traiter.

— Les *attaques à clairs connus*, pour lesquelles l'attaquant possède des chiffrés mais aussi les clairs correspondants (ce scénario intervient notamment pour les cryptanalyses linéaires).

— Les *attaques à clairs choisis* correspondent au cas où l'attaquant demande le chiffrement d'un ensemble de messages qu'il choisit. Ce scénario interviendra notamment dans la plupart des attaques différentielles. On peut aussi imaginer un modèle où l'attaquant demande au fur et à mesure quels clairs il veut voir chiffrer, en fonction de l'information qu'il aura déduite des premiers chiffrés. On parlera alors d'attaque à clairs choisis adaptatifs.

— Les *attaques à chiffrés choisis*, qui sont une variante de l'attaque précédente mais dans laquelle l'attaquant demande le déchiffrement d'un ensemble d'éléments qu'il choisit. Certaines attaques nécessitent la capacité de choisir des chiffrés et des clairs, comme

5. Non seulement par les concepteurs mais aussi par d'autres cryptanalystes.

par exemple les attaques par boomerang [Wag99].

Ces modèles sont le plus souvent considérés dans le *modèle usuel*, c'est-à-dire que la clef est fixée et que l'attaquant n'a aucune influence sur elle.

Un second modèle possible, un peu moins étudié car moins réaliste, est le *modèle à clefs liées*. L'objectif de l'attaquant est toujours de retrouver la clef secrète utilisée, mais il possède une capacité supplémentaire correspondant à la possibilité de demander à ce qu'une clef liée à la première soit utilisée. Nous verrons un exemple d'attaque dans ce modèle au chapitre 5.

Finalement, un modèle encore plus favorable à l'attaquant a été introduit en 2007 par Lars Knudsen et Vincent Rijmen [KR07] : c'est le *modèle à clef connue* ou *Known-key distinguishing attack*. Comme son nom l'indique, ce modèle considère que la clef est connue de l'attaquant ; l'objectif de ce dernier est de mettre en avant une propriété structurelle du chiffrement, c'est-à-dire une propriété que possède le chiffrement et qui ne serait pas présente pour un chiffrement parfait (c'est-à-dire pour une permutation aléatoire). Une variante de ce modèle laisse à l'attaquant la liberté de choisir la clef utilisée (*Chosen-key distinguishing attack*).

Ces deux derniers modèles apparaissent moins réalistes mais se justifient néanmoins par des applications concrètes, à commencer par les constructions de fonctions de hachage basées sur des chiffrements par blocs [6]. Dans tous les cas, les attaques construites dans ces modèles mettent en évidence des propriétés indésirables pour un chiffrement que l'on souhaite au plus près d'une permutation aléatoire, et qui sont donc malvenues.

Quel que soit le scénario étudié on considérera toujours que l'attaquant connaît la méthode de chiffrement utilisée, c'est-à-dire qu'il connaît le détail de l'algorithme utilisé pour obtenir le chiffré et que seule la clef lui est inconnue. Cette hypothèse est liée au principe de Kerckhoffs [Ker83], dont l'énoncé est le suivant [7] :

> "Il faut qu'il [le système de chiffrement] n'exige pas le secret, et qu'il puisse sans inconvénient tomber entre les mains de l'ennemi"

Ce principe est soutenu par le fait qu'un système de chiffrement ne doit pas être gardé secret pour permettre son évaluation par le plus grand nombre et gagner la confiance des utilisateurs.

1.3.2 Mesures de l'efficacité d'une cryptanalyse

Dans la plupart des cas, les faiblesses trouvées par les cryptanalystes ne permettent pas d'attaquer la version complète du chiffrement mais une version *affaiblie* de celle-ci, par exemple une version privée de quelques itérations de la fonction de tour dans le cas d'un chiffrement par blocs itératif. La première mesure de la sécurité d'un chiffrement sera alors de déterminer quelle est la version la moins affaiblie possible que l'on arrive à attaquer. Plus la différence entre cette version et la version complète sera faible (c'est-à-dire plus la marge de sécurité sera faible) et moins le chiffrement sera considéré sûr.

Pour évaluer plus précisément encore l'efficacité d'une attaque, on utilisera les trois métriques suivantes :

6. Comme par exemple la construction de Davies-Meyer, pour laquelle le message, que l'adversaire peut contrôler, est utilisé comme clef du chiffrement par blocs.

7. Auguste Kerckhoffs a formulé d'autres critères que celui-ci (par exemple, que le système de chiffrement doit être applicable à la correspondance télégraphique) mais celui-ci est celui qui reste le plus d'actualité bien qu'il remonte à 1883.

1. La complexité en temps, notée C_T, correspondant au nombre total d'opérations réalisées pour retrouver la clef. Elle peut être exprimée soit en nombre d'opérations élémentaires, soit en nombre de chiffrements.
2. La complexité en données, notée C_D, correspondant au nombre de données (chiffrés ou couples clairs/chiffrés) auquel a accès l'attaquant.
3. La complexité en mémoire, notée C_M, exprimant la capacité de stockage requise pour mener à bien l'attaque.

1.4 Nouveaux défis de la cryptographie symétrique

Les débuts de la cryptographie moderne remontent aux années 1970, période durant laquelle un besoin croissant de confidentialité s'est fait sentir du fait du développement des télécommunications. Les connaissances tant en conception qu'en cryptanalyse se sont beaucoup enrichies entre temps, et on compte aujourd'hui un certains nombre de primitives cryptographiques en lesquelles la communauté a confiance (comme par exemple les standards de cryptographie symétrique AES et SHA-3).

Cependant, le développement de nouveaux types de communication numérique, couplé à des évolutions techniques, crée de nouveaux besoins pour lesquels les chiffrements existant ne sont pas adaptés.

Des exemples de besoins émergents pour la cryptographie symétrique sont :

— La cryptographie dédiée aux petits objets connectés, parfois regroupés sous le concept d'Internet des objets (ou *IoT* pour *Internet of Things*). Ceux-ci nécessitent des algorithmes de chiffrement *légers* c'est-à-dire occupant peu de place, consommant peu et/ou s'exécutant rapidement. La branche de la cryptographie s'intéressant à ce problème est appelée cryptographie *à bas coût* ou *légère* (en anglais *lightweight cryptography*).

— Les contextes où la menace d'attaques par canaux auxiliaires est forte, nécessitant de choisir des chiffrements pour lesquels la mise en place d'une contre-mesure se fait sans peine c'est-à-dire permettent de conserver de bonnes performances.

— Les schémas de chiffrement homomorphe hybride, pour lesquels il est important d'avoir une profondeur multiplicative faible.

Nous expliquerons en détails ces problématiques aux sections 3.1, 4.1 et 8.1 respectivement.

Première partie

Cryptanalyse de chiffrements par blocs

Chapitre 2

Introduction aux chiffrements par blocs

Les chiffrements par blocs sont des algorithmes paramétrés par un ensemble de k bits (appelé clef et noté K) transformant les messages clairs par blocs de n bits en des blocs de messages chiffrés de la même taille. Les tailles de bloc les plus répandues sont 64 et 128 bits, mais d'autres formats sont possibles : certains contextes particuliers comme celui de la cryptographie à bas coût comptent des chiffrement acceptant des blocs de 32, 48 ou 96 bits.

De façon plus formelle, on définit un chiffrement par blocs de la façon suivante :

Définition 2.1. *(Chiffrement par blocs) Un chiffrement par blocs est une famille de permutations sur n bits paramétrée par une clef K :*

$$E_K : \mathbb{F}_2^n \to \mathbb{F}_2^n$$

2.1 Modes opératoires

Cette caractéristique de traitement des messages par blocs impose de mettre en place des techniques pour transformer un message clair de taille quelconque en une suite de blocs de taille n, et notamment de savoir gérer les cas où la taille du message n'est pas multiple de la taille d'un bloc : c'est ce qu'on appelle le *padding*. Le cas échéant le dernier bloc se verra complété suivant une règle précise jusqu'à atteindre la taille des n bits nécessaires.

Une fois cela fait, il existe plusieurs méthodes ou *modes* pour traiter cet ensemble de blocs. Le mode le plus naturel est ECB [1] et consiste à chiffrer indépendamment chaque bloc de message (voir figure 2.1). Le défaut évident de cette méthode est que deux blocs de messages identiques donneront des blocs de chiffrés identiques, ce qui pourrait permettre à un attaquant d'obtenir de l'information sur la structure du message (ses répétitions par exemple) à partir de celle des chiffrés.

1. Acronyme de son appellation anglaise *Electronic Code Book*.

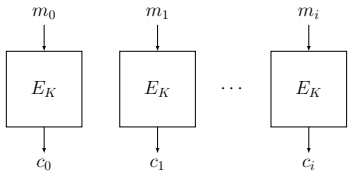


FIGURE 2.1 – Mode ECB : chaque bloc de message m_i est chiffré indépendamment des autres avec le même chiffrement par blocs E_K.

Le mode CBC [2], dont le fonctionnement est représenté figure 2.2 permet d'éviter ce problème puisque chaque bloc de chiffré dépendra de tous les chiffrés précédents.

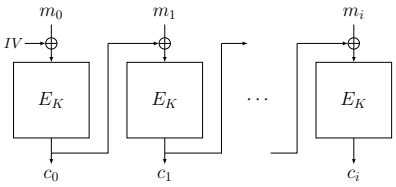


FIGURE 2.2 – Mode CBC : le système est initialisé avec un vecteur d'initialisation public (IV) puis les blocs de message clairs sont traités en chaîne.

Plusieurs autres modes opératoires existent (CFB (Cipher Feedback), OFB (Output Feedback), CTR (counter)...), dont certains comme CTR peuvent être vus comme des techniques permettant d'utiliser un chiffrement par blocs pour construire un chiffrement à flot.

2.2 Principes de conception des chiffrements par blocs

2.2.1 Généralités

Si on se réfère à la définition donnée plus haut, construire un chiffrement par blocs revient à spécifier 2^k familles de permutations de n bits, qui plus est en s'assurant de la sécurité cryptographique apportée par ces permutations. Cette tâche n'est pas chose aisée, surtout compte tenu de la taille des ensembles considérés (le plus souvent $n = k = 128$) et c'est pourquoi la totalité des chiffrements actuels est construite sur le principe du chiffrement itératif. L'idée ici consiste à former l'algorithme par la répétition d'une même fonction [3] appelée fonction de tour, elle-même composée de petites fonctions faciles à analyser. La fonction de tour est itérée suffisamment jusqu'à atteindre le niveau de sécurité voulu.

Pour faire intervenir la clef dans une telle construction on calculera à partir de la clef originale K (appelée clef-maître) un ensemble de *clefs de tours* (aussi appelées *sous-clefs*) qui interviendront dans le calcul des fonctions de tour associées. Cet algorithme de calcul est appelé *cadencement de clefs* ou *dérivation de clefs* [4].

2. Pour *Cipher Block Chaining*.
3. Les itérations peuvent varier par l'utilisation de constantes différentes.
4. En anglais *Key-Schedule algorithm*.

Si on note par F_k la fonction de tour utilisant la clef de tour k, un chiffrement itératif E_K exécutant r tours peut être décomposé de la façon suivante :

$$E_K(M) = F_{k_{r-1}} \circ F_{k_{r-2}} \circ \cdots \circ F_{k_0}(M)$$

Le principe général d'un chiffrement itératif est représenté sur la figure 2.3.

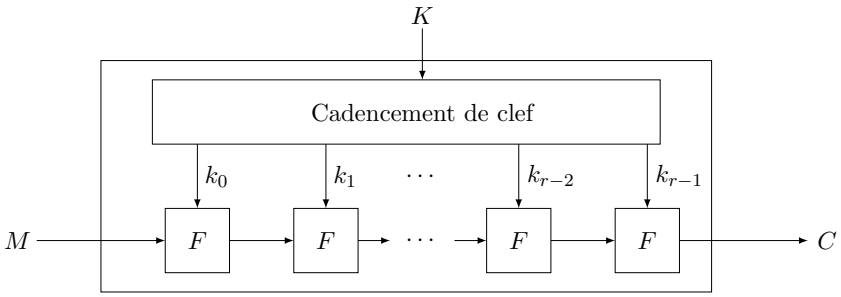


FIGURE 2.3 – Chiffrement par blocs itératif : ce type de chiffrement est construit par la répétition d'une même fonction F paramétrée par une sous-clef k_i calculée à partir de la clef-maître K.

Il existe principalement deux types de constructions de chiffrement itérés : les réseaux de Feistel et les réseaux de substitution-permutation.

2.2.2 Réseaux de Feistel

Cette première construction tient son nom de Horst Feistel, qui l'introduisit en 1974 lors de la conception du chiffrement Lucifer [Fei74], qui inspira entre autre le standard de chiffrement DES [5] [FIP77], usité de 1977 à 2000.

La fonction de tour d'un schéma de Feistel fonctionne de la façon suivante (voir figure 2.4) :

$$\begin{array}{rcl} \mathbb{F}_2^{\frac{n}{2}} \times \mathbb{F}_2^{\frac{n}{2}} & \to & \mathbb{F}_2^{\frac{n}{2}} \times \mathbb{F}_2^{\frac{n}{2}} \\ (L_i, R_i) & \mapsto & (L_{i+1}, R_{i+1}) = (R_i, L_i \oplus f(R_i, k_i)). \end{array} \tag{2.1}$$

L'état interne est formé de deux parties égales, notées L_i et R_i : la moitié de droite est transformée par la fonction f paramétrée par la clef de tour k_i. Le résultat de ce calcul est additionné par `XOR` à la moitié de gauche puis finalement les deux moitiés échangent leurs places respectives avant le début du tour suivant.

Cette construction permet plus de liberté dans le choix de ses composants (dans la définition de f notamment) par rapport aux réseaux de substitution-permutation [6] puisque f n'a pas besoin d'être inversible pour que la fonction de tour le soit.

5. Data Encryption Standard.
6. Voir le paragraphe suivant.

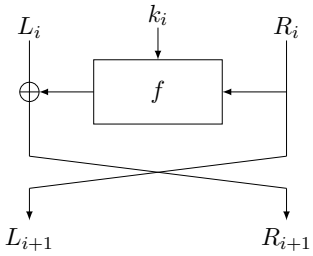


FIGURE 2.4 – Fonction de tour d'un réseau de Feistel

On peut citer les chiffrements LBlock [WZ11], Simon [BSS+15] ou encore RoadRunner [BS15] qui appartiennent à cette catégorie.

Plusieurs généralisations de cette construction existent, à commencer par les réseaux de Feistel généralisés [Nyb96] qui découpent le message en plus de deux parties (on parle alors de réseaux de Feistel à plusieurs branches). Par exemple le chiffrement standardisé par l'ISO-29192 nommé CLEFIA [SSA+07] utilise 4 branches de 32 bits, Twine [SMMK12] utilise 16 branches de 4 bits et Piccolo [SIH+11] utilise 4 branches de 16 bits.

Une autre généralisation possible des réseaux de Feistel originaux consiste à utiliser deux branches de poids différents : c'est ce qu'on appelle les réseaux de Feistel non équilibrés (*Unbalanced Feistel Networks* [SK96]).

2.2.3 Réseaux de substitution-permutation (SPN)

Une autre façon de concevoir un algorithme de chiffrement est d'utiliser un réseau de substitution-permutation, dont le principe est de suivre au plus près les deux critères de conception énoncés par Claude Shannon [Sha49], sous le nom de *confusion* et *diffusion*. Ces deux principes ont été pensés pour apporter une résistance contre les analyses statistiques, et peuvent se décrire comme suit :

confusion : l'algorithme de chiffrement doit rendre la relation entre le clair, la clef et le chiffré la plus complexe possible,

diffusion : l'algorithme de chiffrement doit diffuser l'information contenue dans le clair et la clef dans chaque bit du chiffré.

Comme représenté figure 2.5, la fonction de tour d'un réseau de substitution-permutation est composée (en plus de l'addition de clef habituelle) de deux opérations : l'application de petites substitutions (typiquement de 4 ou 8 bits [7]), appelées *boîtes-S*, apportant la confusion, et le calcul d'une opération linéaire M, apportant la diffusion.

Cette structure ne garantissant pas l'inversibilité, elle impose que chaque composante soit inversible.

Le standard de chiffrement actuel nommé AES suit cette construction.

7. La petite taille de ces substitutions s'explique par deux facteurs principaux : d'un point de vue pratique, l'implémentation d'une fonction non-linéaire de grande taille serait trop coûteux et d'un point de vue mathématique il est plus facile de construire de petites fonctions non-linéaires avec de bonnes propriétés (permettant notamment d'améliorer la résistance du chiffrement à certains types d'attaques).

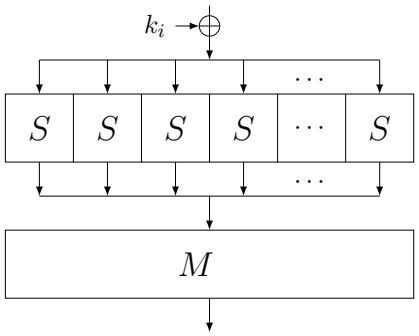


FIGURE 2.5 – Réseau de Substitution-Permutation (SPN)

L'Advanced Encryption Standard (AES)

Face à la menace d'obsolescence du DES dont la taille de clef était devenue trop petite au vu des améliorations technologiques, le NIST [8] décida en 1997 d'organiser une compétition publique pour sélectionner un nouveau standard de chiffrement. Les directives imposées aux constructions candidates étaient de pouvoir supporter des clefs de 128, 192 et 256 bits et d'opérer sur des blocs de 128 bits.

Après 3 ans d'analyse intensive des 15 candidats ce fut le chiffrement Rijndael, conçu par les cryptologues belges Joan Daemen et Vincent Rijmen, qui fut retenu. Cet algorithme opère sur des blocs de 128 bits représentés sous forme d'une matrice carré de 4 octets sur 4 octets. La fonction de tour est composée de 4 opérations (voir figure 2.6) : l'addition de clef (ARK), l'application d'une substitution d'octets (SB) puis de deux opérations linéaires (SR et MC).

Le chiffrement se décline en 3 variantes dont les seules différences sont le nombre d'itérations r de la fonction de tour (10, 12 et 14), la taille de la clef-maître (respectivement 128, 192 et 256 bits) et l'algorithme de cadencement de clef.

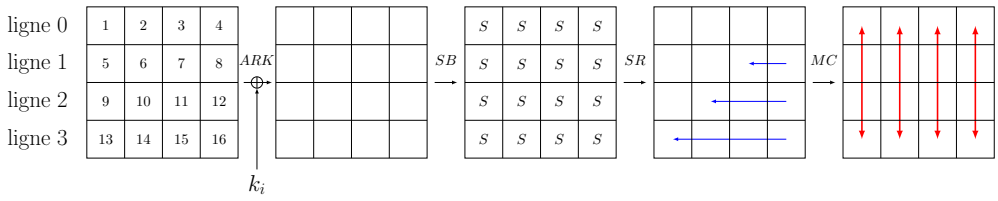


FIGURE 2.6 – Fonction de tour de l'Advanced Encryption Standard (AES)

Quelle que soit la version choisie, les clefs de tour font la même taille que l'état interne (128 bits) ce qui permet d'additionner les deux en début de tour (c'est l'opération dénotée ARK). Ensuite, chaque octet est permuté par la même boîte-S puis chaque ligne i subit une rotation de i octets vers la gauche. Finalement, l'opération MC transforme chaque colonne en la multipliant par la matrice M définie par :

8. National Institute of Standards and Technology, agence de standardisation des États-Unis.

$$M = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix}$$

Pour cette opération chaque octet est vu comme un élément de

$$GF(2^8) = GF(2)[X]/(X^8 + X^4 + X^3 + X + 1).$$

Notons ici que cette matrice a de très bonnes propriétés de diffusion et plus particulièrement qu'elle est associée à un code dit *MDS* (pour *Maximum Distance Separable*). Cette association implique que le nombre total d'octets possédant une différence (sur les 8 octets en entrée et en sortie) vaudra au minimum 5. Par exemple, dans le cas précis où deux messages d'entrée différent sur 1 seul octet, la différence de sortie affectera l'ensemble des 4 octets de l'image.

Le dernier tour ne comporte pas l'opération *MixColumns* et finit par une addition de sous-clef supplémentaire.

2.3 Principales attaques sur les chiffrements par blocs

Comme nous l'avons vu dans l'introduction de cette thèse, évaluer la sécurité d'un chiffrement consiste à étudier au plus près ses propriétés pour tenter de retrouver la clef secrète de façon significativement plus efficace que par force brute. Si, malgré de nombreuses analyses extérieures, il apparaît qu'on ne trouve pas de telles techniques (que l'on appelle des *attaques* ou *cryptanalyses*), le chiffrement est considéré comme sûr.

2.3.1 Bref aperçu des attaques existantes

Attaques statistiques

Le développement de la cryptanalyse date des années 1990, avec notamment la découverte de deux techniques très puissantes que sont la cryptanalyse différentielle [BS90] et la cryptanalyse linéaire [TG91, Mat93] suivies ensuite par de nombreuses variantes. Ces deux techniques sont d'une importance telle qu'elles ont donné lieu à des critères de conceptions permettant de s'en prémunir.

Les cryptanalyses différentielles et linéaires sont des attaques statistiques, c'est-à-dire qu'elles vont exploiter un biais statistique dans le chiffrement par rapport au comportement d'une permutation aléatoire :

— Dans le cas de la cryptanalyse différentielle, ce comportement correspond à un biais statistique dans la distribution de différences observées en sortie du chiffrement [9] lorsque la différence d'entrée entre deux messages clairs est fixée. Ce type d'attaque (et ses variantes) est celui qui nous a permis de cryptanalyser les chiffrements KLEIN, `Zorro` et PICARO. Nous lui consacrons la section 2.3.2.

— Dans le cas de la cryptanalyse linéaire ce comportement correspond à des approximations affines faisant intervenir des bits du texte clair, du chiffré et de la clef qui sont vérifiées avec une probabilité s'écartant significativement de ce qu'on observerait pour

9. Celle-ci sera presque toujours définie par le `XOR` bit à bit entre deux messages.

une permutation aléatoire.

Il existe un grand nombre de variantes des attaques différentielles et linéaires, parmi lesquelles :

— La cryptanalyse différentielle tronquée [Knu94]
— La cryptanalyse différentielle impossible [Knu98, BBS99]
— La cryptanalyse différentielle multiple [BG11]
— La cryptanalyse différentielle d'ordre supérieur [Knu94, Lai94]
— La cryptanalyse par boomerang [Wag99]
— La cryptanalyse différentielle-linéaire [LH94]
— La cryptanalyse linéaire multiple [BCQ04]
— La cryptanalyse linéaire multidimensionnelle [CHN08]
— Les attaques par zéro-corrélation [BR14]

Nous n'allons pas décrire toutes ces attaques ici mais un exemple d'attaque par différentielles tronquées sera donné dans le chapitre 3.

Autres attaques

Les attaques statistiques ne sont pas les seules menaces pesant sur les chiffrements par blocs.

On peut par exemple évoquer les attaques algébriques [CP02], qui consistent à exploiter des relations entre les bits du message clair, du chiffré et de la clef. Pour réussir à retrouver la clef de manière plus rapide que la recherche exhaustive, l'objectif ici sera de trouver des relations suffisamment simples de sorte à être en mesure de former un système d'équations en les bits de la clef qui soit raisonnablement facile à résoudre (car de faible degré, de petite taille ou très structuré). Ce type d'attaques est très efficace dans le cas des chiffrements à flot mais peut aussi s'avérer intéressant pour les chiffrements par blocs, notamment lorsqu'elles sont combinées à d'autres attaques (voir par exemple l'attaque différentielle algébrique de Martin Albrecht et Carlos Cid [AC09] ou notre algorithme de recouvrement de clef sur les PSPN au chapitre 4).

Donnons comme dernier exemple les attaques par *rencontre par le milieu* [DH77] (ou Meet-in-the-Middle attacks). Ce type d'attaques requiert, comme le précédent, la connaissance de messages clairs et des messages chiffrés correspondants. L'idée est de calculer une même partie d'état interne (correspondant le plus souvent à l'état interne obtenu après le passage d'environ la moitié des tours du chiffrement itératif) de deux façon différentes : à partir du message clair en chiffrant jusqu'à l'état interne, et à partir du chiffré en déchiffrant jusqu'à ce même état. Si la valeur de la partie de l'état du milieu ne concorde pas, on en déduira que les hypothèses faites sur la clef pour chiffrer et déchiffrer sont mauvaises. Dans le cas contraire les parties de clef seront considérées comme probables et traitées plus avant.

Un bon panorama des attaques classiques existant sur les chiffrements par blocs peut être trouvé dans [SPQ03].

2.3.2 Cryptanalyse différentielle

Cette section donne une description plus détaillée des attaques différentielles sur les chiffrements par blocs itératifs. Dans toute la section nous raisonnons sur un chiffrement par blocs itératif E comptant r tours, agissant sur des blocs de n bits et utilisant une clef K

de k bits. Une description plus complète de la théorie des attaques différentielles et de leurs généralisations peut être trouvée dans la thèse de Céline Blondeau [Blo11].

Comme nous l'avons vu dans la section précédente, la cryptanalyse différentielle étudie la propagation des différences et plus précisément cherche à mettre en avant des comportements du chiffrement qui s'éloignent de ceux d'une permutation aléatoire. La notion de base apparaissant dans cette analyse est celle de *différentielle*.

Définition 2.2. *(Différentielle) Une différentielle sur t tours d'un chiffrement itératif par blocs est un couple $(\delta_0, \delta_t) \in \mathbb{F}_2^n \times \mathbb{F}_2^n$ formé d'une différence en entrée et d'une différence en sortie après t tours.*

On notera E^t le chiffrement E réduit aux t premiers tours.

Pour les cryptanalyses différentielles simples, l'objectif sera de trouver une différentielle de probabilité élevée. La probabilité d'une différentielle est définie de la façon suivante :

Définition 2.3. *(Probabilité d'une différentielle) La probabilité d'une différentielle (δ_0, δ_t), notée $P(\delta_0 \to \delta_t)$ est définie par :*

$$P(\delta_0 \to \delta_t) = P_{X,K}(E_K^t(X) \oplus E_K^t(X \oplus \delta_0) = \delta_t),$$

où $P_{X,K}$ est la probabilité calculée en moyenne sur tous les messages $X \in \mathbb{F}_2^n$ en entrée et sur toutes les clefs $K \in \mathbb{F}_2^k$ possibles.

Dans la pratique, il sera très difficile de calculer la probabilité d'une différentielle sur t tours du chiffrement lorsque t devient grand. On choisira plutôt de s'intéresser aux caractéristiques, c'est-à-dire que l'on spécifiera l'évolution de la différence, tour après tour.

Définition 2.4. *(Caractéristique différentielle) Une caractéristique différentielle sur t tours d'un chiffrement E est un $t+1$-uplet $(\delta_0, \delta_1, \cdots, \delta_t) \in (\mathbb{F}_2^n)^{t+1}$ spécifiant la différence en entrée et les différences à la fin de chaque tour.*

À clef secrète fixée, on parlera de *bonne paire* (ou de paire conforme) pour désigner deux messages d'entrée dont les différences générées sont celles données par la caractéristique et de *mauvaise paire* dans le cas contraire. Une caractéristique différentielle est parfois aussi appelée *chemin différentiel*. Sa probabilité est définie de la façon suivante :

Définition 2.5. *(Probabilité d'une caractéristique différentielle) La probabilité d'une caractéristique différentielle $(\delta_0, \delta_1, \cdots, \delta_t) \in (\mathbb{F}_2^n)^{t+1}$ est donnée par :*

$$P(\delta_0, \delta_1, \cdots, \delta_t) = P_{X,K}(E_K^i(X) \oplus E_K^i(X \oplus \delta_0) = \delta_i, \forall i \in [1, t])$$

Pour pouvoir calculer une probabilité théorique, on émet l'hypothèse que le chiffrement est de Markov [LMM91], ce qui implique que la probabilité en moyenne sur les clefs d'une caractéristique est égale au produit des probabilités des différentielles sur un tour.

Définition 2.6. *(Chiffrement de Markov) Un chiffrement par blocs itératif E est dit de Markov relativement à la cryptanalyse différentielle si la probabilité d'observer une différence en sortie connaissant la différence en entrée est indépendante de la clef utilisée pour chiffrer.*

En admettant de plus que les clefs de tour sont indépendantes et uniformément distribuées dans l'espace des clefs, on obtient que la probabilité d'une caractéristique vaut le produit des probabilités des différentielles sur un tour qui la composent.

Ainsi, dans la pratique, un attaquant étudiera la fonction de tour du chiffrement pour en déduire des caractéristiques de forte probabilité sur un tour, puis les combinera entre elles pour former une caractéristique différentielle. Il calculera sa probabilité en réalisant le produit des probabilités des caractéristiques sur un tour.

Une différentielle est bien entendu formée de nombreuses caractéristiques. Dans la plupart des cas, l'une d'elles aura une probabilité beaucoup plus forte que les autres et donnera une bonne approximation de la probabilité de la différentielle.

Recherche de caractéristiques

Pour calculer la probabilité d'une différentielle sur un tour, on étudie son évolution à travers les différentes opérations de la fonction de tour F. Ce calcul est trivial pour toutes les opérations linéaires L de F puisque pour celles-ci la différence évolue toujours de δ à $L(\delta)$. Au contraire, l'évolution d'une différence non nulle par passage dans une opération non-linéaire ne sera pas prédictible avec certitude si seule la différence d'entrée et connue et non les valeurs.

L'attaquant va alors raisonner avec des probabilités. Un outil utile dans ce cas est la *table de distribution des différences* (que l'on abrégera en *DDT* pour *Difference Distribution Table* dans la suite de ce document).

Définition 2.7. *(Table de distribution des différences) La table de distribution des différences d'une boîte-S* S *de* η *bits sur* η *est une matrice de taille* $2^\eta \times 2^\eta$ *telle que*

$$\mathsf{DDT}[a,b] = \#\{X \in \mathbb{F}_2^\eta \mid \mathsf{S}(X \oplus a) \oplus \mathsf{S}(X) = b\}.$$

Cette table fournit à la ligne a, colonne b le nombre de valeurs $X \in \mathbb{F}_2^\eta$ permettant la transition de la différence a à la différence b par la boîte-S. Pour obtenir la probabilité correspondante, on divisera cette valeur par le nombre d'entrées X possibles :

$$P(a \to_{\mathsf{S}} b) = \frac{\mathsf{DDT}[a,b]}{2^\eta}$$

Le principe du calcul de la probabilité d'une caractéristique est récapitulé à la figure 2.7.

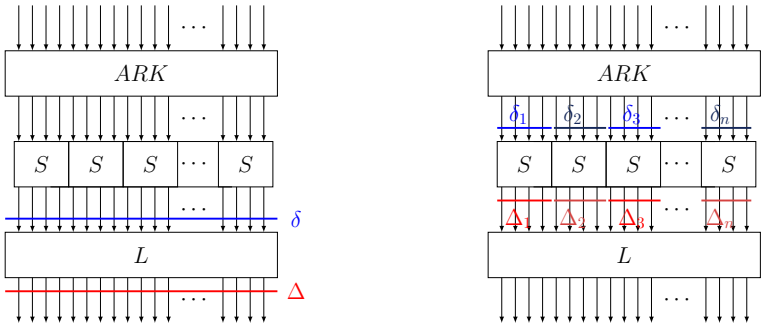


FIGURE 2.7 – Calcul de probabilité d'une caractéristique sur un tour d'un SPN : le passage d'une différence de δ à Δ par l'opération linéaire L se fait avec probabilité 1 si $\Delta = L(\delta)$, avec probabilité nulle sinon. La probabilité d'une transition par l'étape non-linéaire est donnée par le produit des transitions $\delta_i \to_{\mathsf{S}} \Delta_i$ de chacune des boîtes-S. Pour déterminer celui-ci on se réfère à la DDT de S.

Pour obtenir une caractéristique de probabilité élevée, l'attaquant cherchera à minimiser le nombre de boîtes-S ayant des différences d'entrée non nulles [10] et à faire en sorte que leurs probabilités de transition soient le plus élevé possible. Cette probabilité de transition maximale est dérivée de l'uniformité différentielle de la boîte-S.

Définition 2.8. *(Uniformité différentielle) L'uniformité différentielle d'une boîte-S* S *est le plus grand coefficient présent dans la* DDT *privée de sa première ligne et de sa première colonne.*

Comme nous allons le voir dans la description de l'attaque sur le chiffrement KLEIN au chapitre 3, les attaques par différentielle tronquée ne s'intéressent pas au détail précis des transitions effectuées par les boîtes-S mais seulement à leur activité ou inactivité. Dans ce cas-ci, les étapes de diffusion du chiffrement auront un rôle primordial.

Si un attaquant réussit à trouver une différentielle de forte probabilité, il sera en mesure de construire un distingueur :

Définition 2.9. *(Distingueur) Un distingueur est un algorithme qui par un jeu de questions/réponses à une permutation arrive à décider avec probabilité supérieure à* $\frac{1}{2}$ *s'il s'agit d'une permutation aléatoire ou s'il s'agit d'un chiffrement particulier.*

Pour une permutation aléatoire, toute différence de sortie sur n bits a une probabilité d'apparition égale à 2^{-n}, quelle que soit la différence d'entrée. Si on met en avant une différentielle sur t tours (δ_0, δ_t) de E apparaissant avec probabilité p significativement supérieure à 2^{-n}, on sera en mesure de les distinguer. Par exemple, on pourra demander le chiffrement d'un multiple de p^{-1} paires de messages de différence δ_0 et observer les différences obtenues. Si le nombre de paires demandées est inférieur à 2^n, une permutation aléatoire renverra tout au plus quelques fois la différence δ_t, alors que le nombre d'apparition de δ_t sera environ égal au multiple choisit si les chiffrés ont été produits par E^t.

En plus d'indiquer une faiblesse du chiffrement, un distingueur peut être mis à profit pour monter des attaques, par exemple avec la technique dite de l'*attaque sur le dernier tour.*

10. Ces boîtes-S sont dites *actives* en opposition aux boîtes-S *inactives* dont la différence d'entrée et nulle et dont la transition est de probabilité 1.

Exemple d'exploitation d'une différentielle de forte probabilité : les attaques sur le dernier tour

Les attaques sur le dernier tour permettent d'utiliser un distingueur statistique pour retrouver la clef du chiffrement. Ce distingueur peut être de différentes natures : ici, nous nous concentrons sur les attaques différentielles, mais il faut garder à l'esprit qu'il est aussi possible d'exploiter d'autres distingueurs, linéaires par exemple.

Le cas d'une attaque différentielle sur le dernier tour est représenté figure 2.8 : l'attaquant possède une différentielle notée (δ, Δ) de forte probabilité p, positionnée entre le premier tour et l'avant-dernier tour. Il demande le chiffrement d'un nombre suffisant de paires de messages dont la différence d'entrée vaut δ puis réalise une hypothèse sur la valeur de (certains bits de) la clef du dernier tour (notée $\overline{k_{r-1}}$ sur la figure) de sorte à réussir à déchiffrer partiellement les paires de chiffrés et à observer la différence en ce point, qu'il compare à Δ. Si, lorsqu'il calcule la fréquence d'apparition de Δ, il retrouve le biais, c'est-à-dire si cette fréquence est égale à celle prédite, il conclura que son hypothèse sur $\overline{k_{r-1}}$ est correcte.

Cette attaque repose sur le principe dit de *répartition aléatoire par fausse clef*, qui établit qu'une mauvaise hypothèse n'amènera pas de biais et au contraire renverra un comportement très proche du comportement aléatoire. Nous le définissons ici dans le cas des attaques différentielles, avec les notations utilisées précédemment et dans la figure 2.8 :

Définition 2.10. *(Hypothèse de répartition aléatoire par fausse clef) Soit $E_K : \mathbb{F}_2^n \to \mathbb{F}_2^n$ un chiffrement par blocs paramétré par la clef K et de fonction de tour F. On note k_{r-1} la clef de tour du dernier tour. L'hypothèse de répartition aléatoire par fausse clef consiste à supposer que :*

$$P_X(F^{-1}_{\overline{k_{r-1}}}(E_K(X)) \oplus F^{-1}_{\overline{k_{r-1}}}(E_K(X \oplus \delta)) = \Delta) = \begin{cases} p & si\ \overline{k_{r-1}} = k_{r-1} \\ \frac{1}{2^n - 1} & sinon \end{cases}$$

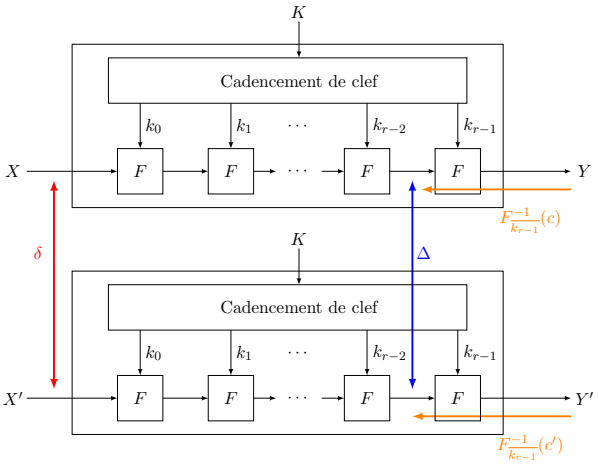


FIGURE 2.8 – Schématisation d'une attaque différentielle sur le dernier tour.

Chapitre 3

Cryptanalyse différentielle tronquée de la famille de chiffrements KLEIN

Ce chapitre présente le travail réalisé avec María Naya-Plasencia sur la famille de chiffrements à bas coût KLEIN [GNL11] qui donna lieu à l'article *Cryptanalysis of KLEIN* paru à FSE 2014 [LN14].

Alors que les meilleurs résultats des cryptanalyses précédentes n'attaquaient que 10 tours sur les 12 que comprend la plus petite instance de cette famille, nos travaux exhibent la première cryptanalyse du chiffrement complet. Nous montrons aussi que nous pouvons étendre nos observations aux 2 autres instances de la famille (utilisant des clefs plus longues et itérant plus de tours) ce qui nous permet d'obtenir les meilleures attaques sur ces versions.

Nos résultats sont appuyés par plusieurs vérifications expérimentales de l'attaque sur des versions réduites du chiffrement.

3.1 Description de KLEIN et cryptanalyses précédentes

3.1.1 Cryptographie à bas coût

La cryptographie à bas coût s'attache à fournir la sécurité nécessaire à des applications disposant de très peu de ressources, présentes sur des supports comme les puces RFID [1], les nœuds des réseaux de capteurs sans fil ou les cartes sans contact.

Ce besoin pressant d'algorithmes de chiffrement à bas coût s'explique par la tendance actuelle de prolifération des petits objets connectés, offrant un large panel d'applications (plus ou moins indispensables) allant de l'optimisation de la gestion des chaînes logistiques aux frigos interagissant avec leur contenu pour avertir leurs propriétaires des produits périmés ou manquants.

Les chiffrements existants ne sont pas adaptés aux besoins de sécurisation de ces applications, ce qui impose de construire spécifiquement de nouveaux chiffrements. De nombreuses primitives ont été proposées récemment [2], parmi lesquelles PRESENT [BKL+07],

1. Pour *Radio Frequency IDentification* ou puces de radio-identification.
2. Une description des principaux chiffrements par blocs et à flot à bas coût peut être trouvée sur le portail CryptoLUX [BP15] du groupe de Cryptographie de l'université du Luxembourg : https://www.cryptolux.org/index.php/Lightweight_Block_Ciphers et https://www.cryptolux.org/index.php/Lightweight_Stream_Ciphers.

CLEFIA [SSA+07], LBlock [WZ11], TWINE [SMMK12], PRINTCipher [KLPR10], KATAN et KTANTAN [CDK09], SIMON et SPECK [BSS+15], LED [GPPR11], ainsi que la famille de chiffrements que nous étudions dans ce chapitre, appelée KLEIN [GNL11] .

Il est important de noter que ces chiffrements à bas coût ne poursuivent pas tous les mêmes objectifs et qu'ils répondent à des contraintes différentes selon la finalité visée. La première catégorisation pouvant être faite est la différentiation entre les primitives aspirant à des intégrations sur des supports matériels ou logiciels.

— Pour les chiffrements visant des usages logiciels (*sofware-oriented*), les paramètres qu'un concepteur cherchera à optimiser sont : la mémoire nécessaire pour mener le calcul (RAM) et celle nécessaire pour stocker l'algorithme (par exemple sur une ROM), la rapidité d'exécution (nombre de cycles d'horloge nécessaires pour traiter une unité de donnée) ou encore la latence.

— Pour les chiffrements visant des usages matériels (*hardware-oriented*), donc une utilisation sur des supports comme les puces RFID, la mémoire nécessaire pour mener le calcul ainsi que la taille du circuit nécessaire au stockage de l'algorithme s'exprimera en terme de *Gate Equivalent (GE)*[3], une unité servant à mesurer la taille du circuit électronique. En plus de ces quantités, il pourra être nécessaire d'optimiser le temps d'exécution et la latence de l'algorithme, ainsi que la puissance qu'il requiert.

Selon les caractéristiques précises de l'application, il pourra être plus important d'optimiser l'une ou l'autre de ces quantités. Pour ne donner que quelques illustrations de ceci, citons le chiffrement PRINCE [BCG+12] visant une implémentation matérielle pour des applications comme le chiffrement de données de disques SSD, et par suite ayant été conçu pour optimiser sa latence. Les concepteurs du chiffrement Midori [BBI+15] se sont quant à eux focalisés sur l'optimisation de l'énergie consommée par bit lors du chiffrement et du déchiffrement et mentionnent comme applications visées les implants médicaux et les nœuds de capteurs.

Les choix que réalisera le concepteur pour répondre à ces contraintes dépendront aussi des caractéristiques précises de la plateforme considérée et des opérations disponibles sur celles-ci.

3.1.2 Description de la famille de chiffrements KLEIN

La famille de chiffrements par blocs KLEIN a été introduite par Zheng Gong, Svetla Nikova et Yee-Wei Law et présentée à la conférence RFIDSec en 2011 [GNL11]. L'objectif visé par ses auteurs était de concevoir un chiffrement à bas coût avec de bonnes performances logicielles, adapté à des supports tels que les nœuds des réseaux de capteurs. Il s'est avéré que les choix de conception réalisés ont permis d'obtenir des performances matérielles elles aussi avantageuses et une implémentation suffisamment compacte pour en faire un candidat intéressant pour des applications matérielles type puces RFID.

La famille KLEIN compte 3 chiffrements agissant tous sur des messages de 64 bits mais différant par la taille de la clef-maître (64, 80 ou 96 bits) ainsi que par le nombre d'itérations de la fonction de tour (respectivement 12, 16 et 20). Ces 3 variantes sont référencées par

3. 1 GE correspond à la surface occupée par une porte NAND.

KLEIN-64, KLEIN-80 et KLEIN-96.

Étant donné que la plus petite version ne possède que 64 bits de clef, son usage n'est pas recommandé par les auteurs pour faire du chiffrement de données. En effet, on considère communément que la taille de clef minimale à considérer est de 80 (voire 100) bits, ce qui étant donnée la puissance de calcul actuelle, permet de se prémunir des recherches exhaustives (ou plus généralement des attaques par compromis temps-mémoire-données). KLEIN-64 est donc uniquement recommandé pour un usage dans une *fonction de hachage* ou un *code d'authentification de message (MAC)*. KLEIN-80 et KLEIN-96 sont recommandés pour le chiffrement (sans restriction sur le mode opératoire, notamment il n'est pas nécessaire de se restreindre uniquement à des chiffrements en préférant un mode comme le mode compteur car le déchiffrement est lui aussi efficace).

Fonction de tour

La fonction de tour de KLEIN suit une structure en réseau de substitution-permutation (SPN). Sa schématisation est donnée à la figure 3.1. Elle possède de grandes similitudes avec l'AES, mais ici — étant donné que le message d'entrée fait 64 bits au lieu de 128 bits pour l'AES — l'unité de base n'est pas l'octet mais le quartet (*nibble* en anglais), défini comme un ensemble de 4 bits. Les noms des fonctions composant la fonction de tour font justement référence à cette spécificité. Plus précisément, la fonction de tour se décompose en 4 opérations :

***AddRoundKey* (ARK)** : Addition bit à bit de la clef de tour (de 64 bits) avec l'état interne.

***SubNibbles* (SN)** : L'état interne est découpé en quartets et chacun est modifié par la boîte-S décrite à la table 3.1.

***RotateNibbles* (RN)** : L'état entier subit une rotation vers la gauche de 16 bits.

***MixNibbles* (MN)** : L'état est découpé en 2 moitiés de 32 bits et chacune est modifiée par la fonction *MixColumns* de l'AES (décrite à la section 2.2).

La fonction de tour est itérée r fois où r vaut 12, 16 et 20 respectivement pour KLEIN-64, KLEIN-80, KLEIN-96.

Notons ici que les concepteurs précisent que leurs choix permettent d'optimiser l'implémentation en considérant des algorithmes orientés octets. C'est une des raisons pour laquelle ils ont préféré l'opération *MixColumns* à une application linéaire orientée quartets (mais qui aurait pu assurer une meilleure diffusion).

Boîte-S

La boîte-S sur 4 bits utilisée dans KLEIN est une involution, ce qui permet d'économiser le codage de son inverse et par conséquent de ne protéger qu'une boîte-S contre les attaques par canaux auxiliaires.

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
S[x]	7	4	a	9	1	f	b	0	c	3	2	6	8	e	d	5

TABLE 3.1 – Boîte-S 4×4 de KLEIN

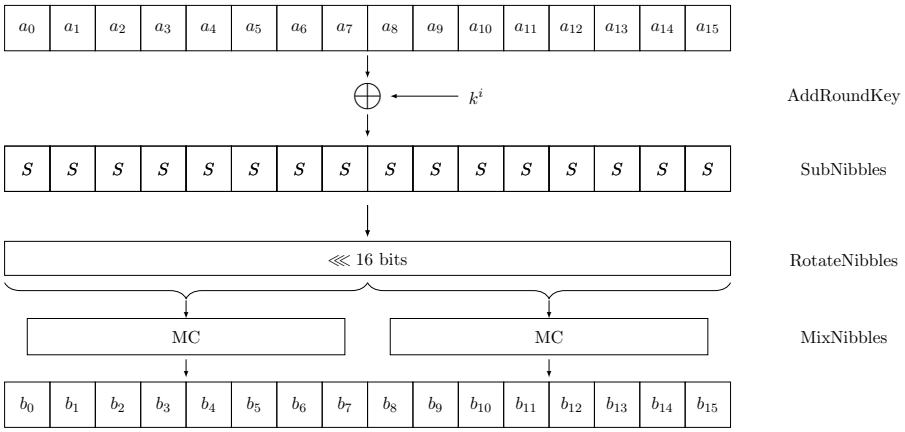


FIGURE 3.1 – Fonction de tour de KLEIN (i-ème itération où $1 \leq i \leq r$) : l'état interne fait 64 bits. Chaque carré représente un quartet, soit 4 bits.

On peut remarquer que la boîte-S n'a pas de point fixe. Si on note la décomposition binaire de l'entrée de la boîte-S $x \in \mathbb{F}_2^4$ par $x = x_3x_2x_1x_0$ (on note x_0 le bit de poids faible) et sa sortie $S(x) = y \in \mathbb{F}_2^4$ par $y = y_3y_2y_1y_0$ alors on a :

$$\begin{aligned}
y_0 &= 1 + x_0 + x_1 + x_3 + x_0x_2 + x_1x_2 + x_1x_3 + x_0x_1x_2 + x_0x_1x_3 \\
y_1 &= 1 + x_0 + x_2 + x_3 + x_1x_2 + x_1x_3 + x_2x_3 + x_0x_1x_3 \\
y_2 &= 1 + x_1 + x_2 + x_0x_2 + x_1x_2 + x_0x_3 + x_0x_1x_2 + x_0x_2x_3 + x_1x_2x_3 \\
y_3 &= x_1 + x_3 + x_0x_2 + x_0x_3 + x_0x_1x_3 + x_1x_2x_3.
\end{aligned}$$

Son uniformité différentielle est de 4, tout comme sa non-linéarité. Les concepteurs de KLEIN ont pris les précautions supplémentaires suivantes. Notons wt le poids de Hamming d'un nombre binaire, Δ_I et Δ_O des différences sur 4 bits et a et b deux masques linéaires :

1. Si $wt(\Delta_I) = wt(\Delta_O) = 1$ alors
$$\#\{x \in \mathbb{F}_2^4 | S(x) \oplus S(x \oplus \Delta_I) = \Delta_O\} \leq 2$$
2. Si $wt(a) = wt(b) = 1$ alors
$$|\sum_{x \in \mathbb{F}_2^4} (-1)^{b \cdot S(x) + a \cdot x}| \leq 4$$

Ces deux propriétés permettent d'assurer une certaine diffusion au niveau des boîtes-S : si la différence d'entrée ne porte que sur 1 bit, alors la probabilité qu'elle ne porte que sur 1 bit en sortie sera très faible (et de même dans le cas des biais linéaires). Les auteurs précisent que si on impose que la boîte-S soit involutive il n'est pas possible d'atteindre la propriété suivie par la boîte-S de PRESENT [BKL+07], à savoir :

$$\text{Si } wt(\Delta_I) = wt(\Delta_O) = 1 \text{ alors } \#\{x \in \mathbb{F}_2^4 | S(x) \oplus S(x \oplus \Delta_I) = \Delta_O\} = 0.$$

Avant d'expliquer le fonctionnement du cadencement de clef, introduisons ici la notion de quartet bas et de quartet haut. On rappelle que l'on note la décomposition binaire d'un octet $a \in \mathbb{F}_2^8$ par $a = a_7a_6a_5a_4|a_3a_2a_1a_0$ où a_0 est le bit de poids faible et a_7 le bit de poids fort.

Définition 3.1. *(Quartets haut et quartet bas) On appellera* ***quartet haut*** *le demi-octet de 4 bits de plus haut poids, soit* $a_7a_6a_5a_4$ *et on désignera par* ***quartet bas*** *les 4 bits de plus petit poids :* $a_3a_2a_1a_0$.

Cadencement de clef

L'algorithme de cadencement de clefs de KLEIN suit une structure de réseau de Feistel, comme représenté à la figure 3.2. L'algorithme calcule dans un premier temps les clefs K^i de la façon décrite dans l'algorithme 1 (celles-ci sont de la même taille que la clef-maître) puis les clefs de tour k^i en sont déduites par troncation aux 64 premiers bits.

Sur la figure et dans l'algorithme on note r le nombre de tours (12, 16 ou 20 selon la version). K^i_b désigne le b-ième octet de la sous-clef[4] du tour i ($i \in [1, r]$). On distingue les deux quartets de K^i_b en notant $K^i_{b,0}$ le quartet bas et $K^i_{b,1}$ le quartet haut et on notera u le nombre d'octets de la clef-maître (donc 8, 10 ou 12 pour KLEIN-64, KLEIN-80 et KLEIN-96 respectivement).

Données : Clef-maître K
Résultat : Clefs K^i, $i \in [1, r]$
$K^1 \leftarrow K$
pour *i=2 à* $r+1$ **faire**
  pour *b=0 à* $\frac{u}{2}-1$ **faire**
    $K^i_b \leftarrow K^{i-1}_{((b+1) \mod \frac{u}{2})+\frac{u}{2}}$
    $K^i_{b+\frac{u}{2}} \leftarrow K^{i-1}_{((b+1) \mod \frac{u}{2})+\frac{u}{2}} \oplus K^{i-1}_{((b+1) \mod \frac{u}{2})}$
  fin
  $K^i_2 \leftarrow K^i_2 \oplus (i-1)$
  $K^i_{\frac{u}{2}+1,0} \leftarrow S[K^i_{\frac{u}{2}+1,0}]$
  $K^i_{\frac{u}{2}+1,1} \leftarrow S[K^i_{\frac{u}{2}+1,1}]$
  $K^i_{\frac{u}{2}+2,0} \leftarrow S[K^i_{\frac{u}{2}+2,0}]$
  $K^i_{\frac{u}{2}+2,1} \leftarrow S[K^i_{\frac{u}{2}+2,1}]$
fin

Algorithme 1 : Algorithme de cadencement de clefs de KLEIN

3.1.3 Résultats précédents et premières propriétés

Nous donnons ici un bref aperçu des travaux précédant les nôtres, et nous nous attardons sur les propriétés qui seront utiles par la suite.

Dans [YWLZ11], Xiaoli Yu et ses coauteurs étudient la résistance de KLEIN aux attaques différentielles tronquées et intégrales. Leurs meilleures attaques atteignent 8 tours de KLEIN-64 et de KLEIN-80. La meilleure attaque précédant la nôtre (qui ne soit pas une recherche exhaustive) est une attaque de Ivica Nikolic et ses coauteurs [NWW15] atteignant 10, 11 et 13 tours de KLEIN-64, KLEIN-80 et KLEIN-96 respectivement. La technique qu'ils introduisent

4. Plus précisément la sous-clef réellement utilisée dans la fonction de tour n'est formée que des 64 premiers bits de K^i.

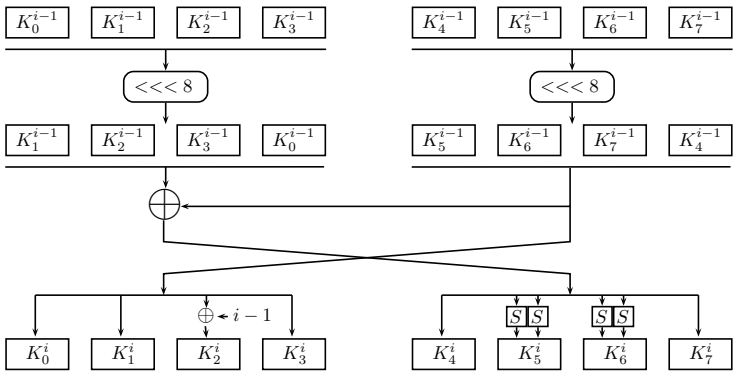


FIGURE 3.2 – Algorithme de cadencement de clef de KLEIN-64 : calcul de K^i à partir de K^{i-1}. Le principe est similaire pour les versions avec des clefs plus longues : l'unique changement proviendra du nombre d'octets sur chaque branche du réseau de Feistel (5 pour KLEIN-80 et 6 pour KLEIN-96 ; les applications de boîtes-S resteront sur le second et troisième octet de la branche droite).

ici est une variante intéressante du MITM qu'ils baptisent attaque en *parallel-cut meet-in-the-middle* (PC MITM).

Les types d'attaques ainsi que leurs complexités sont résumés à la table 3.2.

Version	Référence	tours	C_D	C_T	C_M	type d'attaque
KLEIN-64	[YWLZ11]	7	$2^{34.3}$	$2^{45.5}$	2^{32}	intégrale
	[YWLZ11]	8	2^{32}	$2^{46.8}$	2^{16}	différentielle
	[ANS11]	8	2^{35}	2^{35}	-	différentielle
	[NWW15]	10	1	2^{62}	2^{60}	PC MITM
	[ASA15]	12	2^{39}	$2^{62.84}$	$2^{4.5}$	biclique
	Sec. 3.3	12	$2^{54.5}$	$2^{57.07}$	2^{16}	différentielle
KLEIN-80	[YWLZ11]	8	$2^{34.3}$	$2^{77.5}$	2^{32}	intégrale
	[NWW15]	11	2	2^{74}	2^{74}	PC MITM
	Sec. 3.3	13	2^{52}	2^{76}	2^{16}	différentielle
	[AFL+12]	16	2^{48}	2^{79}	2^{60}	biclique
KLEIN-96	[NWW15]	13	2	2^{94}	2^{82}	PC MITM
	Sec. 3.3	14	$2^{58.4}$	$2^{89.2}$	2^{16}	différentielle
	[AFL+12]	20	2^{32}	$2^{95.18}$	2^{60}	biclique

TABLE 3.2 – Complexités des cryptanalyses précédentes et de nos nouveaux résultats sur KLEIN.

Ces attaques tirent toutes parti de la diffusion très lente entre quartets bas et quartets hauts de la fonction de tour de KLEIN ; cette propriété permet soit de réaliser une différentielle tronquée de forte probabilité, soit de diviser l'état en deux sous-chiffrements (comme dans le *Parallel-Cut Meet-In-The-Middle* [NWW15]).

Propriétés de la fonction de tour

Plus précisément, il est facile de voir que toutes les opérations de la fonction de tour à l'exception de *MixNibbles* sont *nibble-wise*, c'est-à-dire préservent l'indépendance entre chacun des demi-octets de l'état interne [5]. De plus, la diffusion apportée par *MixNibbles* n'est pas très forte, comme le montre la propriété suivante :

Propriété 3.1. *[ANS11, YWLZ11] Soit* $\mathtt{X} \in \mathbb{F}_2^4$ *un quartet de valeur quelconque. Si la différence de 32 bits en entrée de MixColumns est de la forme* `OXOXOXOX` *alors la différence de sortie sera de la même forme avec probabilité* 2^{-3}.

La preuve est donnée en annexe A.

Cette propriété peut être étendue pour construire une caractéristique différentielle tronquée *itérative* sur 1 tour : supposons que la différence d'entrée du tour ne porte que sur les quartets bas de l'état interne de 64 bits. L'évolution de la différence lors du calcul de la fonction de tour se fait de la façon suivante :

- La différence reste inchangée après l'opération d'addition de clef *AddRoundKey*.
- Lorsque l'état est modifié par l'opération *SubNibbles*, chaque quartet est traité indépendamment. La boîte-S étant une bijection, le motif d'activité [6] de l'état reste inchangé.
- L'opération *RotateNibbles* opère ensuite une rotation de l'état interne de 2 octets vers la gauche : comme la valeur du décalage est un multiple de 1 octet, le positionnement relatif quartet haut/ quartet bas est conservé et les quartets hauts restent inactifs.
- La propriété ci-dessus de *MixColumns* implique que les quartets hauts restent inactifs avec probabilité 2^{-6} après application de l'opération *MixNibbles*.

On en déduit la propriété suivante :

Propriété 3.2. *Comme illustré à la figure 3.3, la probabilité qu'une différence d'entrée de tour portant uniquement sur les quartets bas mène à une différence de la même forme en fin de tour vaut* 2^{-6}.

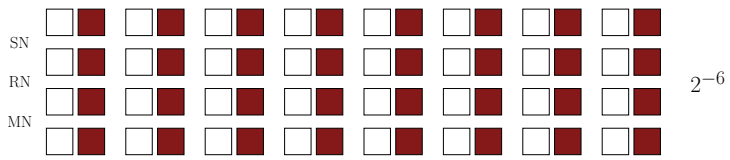


FIGURE 3.3 – Caractéristique différentielle tronquée sur 1 tour de probabilité 2^{-6}.

Remarque 3.1. *Cette propriété reste vraie pour les quartets hauts. Considérons un état avec uniquement les quartets hauts actifs : la forme de la différence sera conservée par les opérations AddRoundKey, SubNibbles et RotateNibbles avec probabilité 1. Comme nous le prouvons en annexe A, nous avons la propriété suivante sur MixNibbles :*

Propriété 3.3. *Une paire ne possédant que des différences sur les quartets hauts aura une différence de la même forme après passage de l'opération MixNibbles avec probabilité* 2^{-6}.

5. Et même la position relative entre quartet haut et quartet bas.
6. C'est-à-dire quels sont les quartets différant entre les deux messages.

Propriétés de l'algorithme de cadencement de clef

Propriété 3.4. *[ANS11, YWLZ11, NWW15] Lors du cadencement de clefs, les quartets bas et hauts ne sont pas mélangés, ce qui implique qu'on peut par exemple calculer les quartets bas d'une clef de tour uniquement à partir des quartets bas de la clef-maître (ou plus généralement des quartets bas de la clef de tour avant qu'elle ne soit tronquée aux 64 premiers bits, notée K^i dans l'algorithme 1).*

Cette propriété implique qu'on peut voir l'algorithme de cadencement de clefs comme deux algorithmes séparés : l'un dérivant tous les quartets bas des sous-clefs à partir des quartets bas de la clef-maître, et l'autre dérivant tous les quartets hauts des sous-clefs à partir des quartets hauts de la clef-maître.

3.1.4 Brève description de l'attaque d'Aumasson et al.

Notre attaque s'inspire beaucoup de la cryptanalyse par différentielles tronquées menée par Aumasson et ses coauteurs dans [ANS11], c'est pourquoi nous la décrivons ici. L'attaque de [ANS11] permet d'attaquer un maximum de 8 tours sur les 12 que comporte KLEIN-64 grâce au chemin différentiel représenté à la figure 3.4. Celui-ci itère la caractéristique de la figure 3.3 du tour 3 au tour 7 et des caractéristiques différentes et de plus forte probabilité dans les 2 premiers tours. La caractéristique est tronquée (donc s'intéresse à l'activité/inactivité des quartets) mais la différence d'entrée (c'est-à-dire la différence des paires de messages sur le quartet 5) est fixée à `0xb` pour assurer que la probabilité de la caractéristique du premier tour soit élevée.

L'attaque sur 8 tours de KLEIN-64 décrite par Aumasson et al. [ANS11] utilise un chemin différentiel sur 7 tours de probabilité p proche de 2^{-32} ainsi qu'une étape de recouvrement de clef proche de celles utilisées dans les attaques sur le dernier tour. L'attaquant commence par demander le chiffrement de p^{-1} paires de messages. Il utilise ensuite la forme spécifique de la différence attendue en fin du tour 7 pour éliminer les paires qui, de façon sûre, ne suivent pas le chemin différentiel. Pour cela il calcule pour chaque paire la différence des chiffrés puis inverse l'opération linéaire *MixNibbles*. Comme il est nécessaire que tous les quartets hauts de la différence obtenue soient inactifs (ce qui correspond à 32 bits nuls, comme représenté à la figure 3.4), la probabilité qu'une différence aléatoire vérifie cette condition sera de 2^{-32} et seules les paires qui suivent effectivement la caractéristique seront retenues.

Chacune des paires restantes permet ensuite d'obtenir de l'information sur la clef. Pour cela, l'attaquant réalisera une hypothèse sur les bits de la sous-clef du dernier tour k^8 nécessaires au calcul des valeurs des quartets bas en sortie de l'opération *SubNibbles*. Cela lui permet de calculer la différence en entrée de l'opération *MixNibbles* du tour 7 et de vérifier si la différence obtenue à ce point possède uniquement les quartets bas d'actifs. Comme les paires considérées sont des paires qui suivent la différentielle, la bonne hypothèse de clef permettra d'obtenir uniquement les quartets bas d'actifs. Au contraire si on observe que des quartets hauts sont actifs on pourra en déduire que l'hypothèse de clef est mauvaise : au total cela nous permettra de réduire l'espace des clefs candidates d'un facteur de 2^{-6}.

L'attaque menée par Aumasson et al. permet de retrouver les 64 bits de clef de KLEIN-64 en répétant cette dernière étape pour plusieurs paires qui suivent la caractéristique. Celles-ci sont obtenues efficacement grâce à la technique dite des *bits neutres*[7], une technique introduite

7. Cette technique permet de construire, à partir des bonnes paires de messages, des nouvelles paires qui

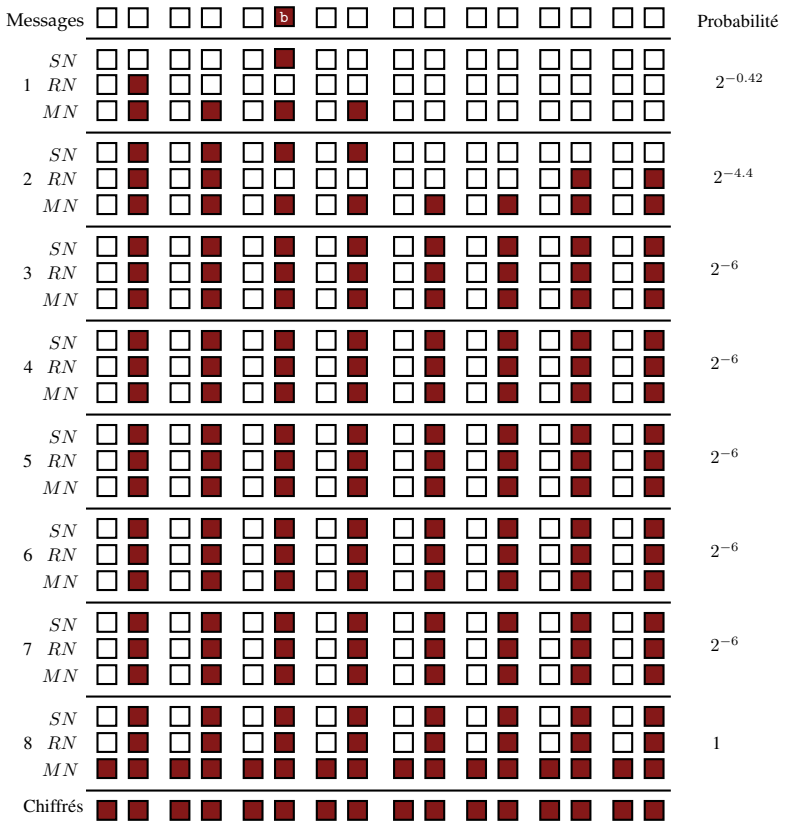


FIGURE 3.4 – Attaque sur 8 tours de KLEIN-64 réalisée dans [ANS11] : le chemin différentiel tronqué est placé des tours 1 à 7 et le dernier tour fait partie de la phase de recouvrement de clef.

par Biham et Chen dans le contexte de la recherche de collision sur SHA-0 [BC04].

3.2 Notre attaque

Notre attaque est une cryptanalyse différentielle tronquée tirant parti d'une caractéristique assez similaire à celle utilisée par Aumasson et al. [ANS11]. Comme nous souhaitons attaquer 4 tours de plus que leur meilleure attaque, la probabilité de notre chemin sera bien inférieure à 2^{-32} et nous ne pourrons plus extraire les paires qui suivent le chemin différentiel simplement en s'assurant que la différence en entrée de la dernière opération *MixNibbles* porte uniquement sur les quartets bas.

L'idée principale de notre attaque consiste à inverser davantage de tours pour pouvoir déterminer les conditions supplémentaires que les paires doivent satisfaire. Grâce à ces conditions, nous serons en mesure de définir des *cribles* nous permettant d'éliminer des paires qui ne suivent pas la caractéristique différentielle et qui par conséquent ne nous permettent pas d'obtenir d'information sur la clef. On utilisera le terme de *facteur de réduction* pour faire référence à la probabilité qu'une paire soit conservée.

suivront la caractéristique avec une probabilité supérieure à la probabilité du cas aléatoire.

Comme nous allons le décrire en détail dans la suite, la mise en place de cette technique nécessite d'associer à chaque couple de messages une (partie de) clef candidate : on travaillera sur des triplets $(C, C', \tilde{k})$ formés par :

— 2 chiffrés C et C' ayant passé le premier crible (de probabilité 2^{-32}),
— une valeur $\tilde{k}$ candidate pour certains bits de la clef.

En inversant plusieurs tours pour chaque triplet (ce qui nécessitera parfois des hypothèses supplémentaires), nous serons en mesure d'identifier et d'éliminer une grande proportion de ceux dont les messages ne suivent pas la caractéristique différentielle.

Si le nombre de paires considéré dans l'attaque est suffisamment élevé, on sera assuré qu'au minimum un triplet formé d'une bonne paire associée à la bonne (partie de) clef se trouvera dans l'ensemble des candidats vérifiant l'ensemble des conditions. On pourra conclure l'attaque en réalisant une hypothèse sur les bits de clef restant et en réalisant des chiffrements tests.

3.2.1 Idée centrale : inverser des tours pour éliminer des mauvaises paires

Le succès de notre attaque repose sur la capacité d'identification de suffisamment de mauvaises paires pour assurer un gain significatif par rapport à la recherche exhaustive. Cette identification est réalisée en inversant en différence[8] des tours et en s'assurant que la différentielle est suivie.

Nous montrons ici qu'inverser un tour t de façon à avoir accès au crible offert par l'opération *MixNibbles* du tour $t-1$ nécessite une hypothèse sur 6 bits mais qu'étant donné le crible auquel cela donne accès (qui a une probabilité de succès valant au maximum 2^{-6}) cette hypothèse résultera dans le pire des cas en un maintien du nombre de candidats.

Rappelons ici que la fonction de tour de KLEIN a comme caractéristique d'apporter très peu de diffusion entre les quartets hauts et bas. Cette propriété est au cœur de notre attaque et implique que, pour obtenir une caractéristique différentielle de forte probabilité, nous considérons des caractéristiques tronquées possédant uniquement des quartets bas actifs.

Notre objectif est d'inverser en différence chacune des paires candidates de façon à pouvoir identifier celles ne suivant pas la caractéristique et à pouvoir les éliminer. Le processus d'inversion décrit ci-dessous doit être réalisé pour chaque paire de messages, l'une après l'autre.

Considérons pour commencer que l'on souhaite inverser en différences une série de tours sur lesquels le chemin différentiel considéré est celui donné à la figure 3.3. La situation de départ est celle représentée à la figure 3.5 : on connaît la valeur de la différence à l'étape A donc aussi à l'étape B (en bleu sur la figure) et cette dernière satisfait le chemin différentiel, c'est-à-dire possède ses quartets hauts inactifs. On souhaite calculer la différence à l'étape F pour déterminer si la paire suit le chemin différentiel, et pouvoir l'éliminer si elle ne s'y conforme pas.

8. Ce qui implique, comme nous allons le montrer, d'inverser en valeur sur les quartets bas.

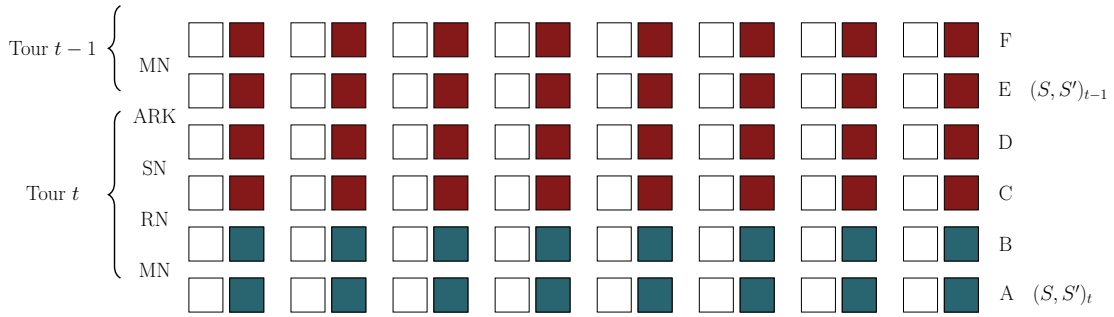


FIGURE 3.5 – Lors d'une étape d'inversion, on souhaite calculer les quartets bas de l'état $(S, S')_{t-1}$ à partir de la valeur des quartets bas de l'état $(S, S')_t$ pour pouvoir accéder à la différence de l'état F.

Nous devons donc inverser en différence les étapes *RotateNibbles*, *SubNibbles*, *AddRoundKey* et *MixNibbles*. La première remarque que nous pouvons faire est que la non-linéarité de l'opération *SubNibbles* nous impose de connaître les *valeurs* des quartets actifs à l'étape C pour pouvoir inverser de façon non probabiliste les boîtes-S. Comme le même problème se répétera lorsqu'on souhaitera inverser l'opération *SubNibbles* du tour précédent, on en déduit qu'il est nécessaire d'inverser en valeur tous les tours sur les quartets bas [9].

La valeur des quartets bas à l'étape C nous permet d'inverser les boîtes-S et de calculer les quartets bas à l'étape D. Pour inverser une étape supplémentaire et obtenir de l'information sur l'état au point E nous devons posséder la valeur des bits de clef correspondants, soit les 32 bits des quartets bas de la clef de tour.

C'est ici que la propriété 3.4 s'avère utile : il n'est pas nécessaire de répéter l'hypothèse de 32 bits à chaque inversion de l'opération *AddRoundKey* mais il suffira de faire une hypothèse sur les quartets bas de la clef-maître pour en déduire les quartets bas de toutes les clefs de tour et donc inverser en quartets bas toutes les opérations *AddRoundKey*.

Une fois cela fait, et comme la fonction *MixNibbles* est linéaire, nous pouvons calculer la valeur de la différence à l'étape F à partir de celle de l'étape E. Le résultat obtenu nous permet de vérifier que le triplet suit la caractéristique (c'est-à-dire possède tous ses quartets hauts inactifs) et de l'éliminer si ce n'est pas le cas. Dans le cas où la caractéristique est suivie à ce tour, on continuera la procédure en s'assurant que le triplet suit aussi la caractéristique au tour précédent.

Pour cela, nous avons besoin de connaître la valeur des quartets bas au point F, ce qui impose d'inverser l'opération *MixNibbles* sur les quartets bas. La difficulté provient du fait que les opérations *MixColumns* n'agissent pas de façon indépendante sur les quartets bas et hauts, donc qu'il est nécessaire de connaître de l'information sur les valeurs des quartets hauts en plus des valeurs des quartets bas que nous possédons.

Comme le montre la propriété suivante, l'information nécessaire est relativement restreinte :

Propriété 3.5. *Pour obtenir les quartets bas résultant de l'inversion de l'opération MixColumns connaissant les quartets bas de l'entrée (e, f, g, h), il est nécessaire de connaître*

9. L'avantage d'une telle procédure est que si une paire se conforme à la différentielle et n'a pas été éliminée lors des inversions de tour, alors on pourra lui faire subir un test supplémentaire correspondant à confronter la valeur des quartets bas obtenus avec ceux du message clair.

3 valeurs dépendant des quartets hauts inconnus :

$$\begin{cases} e_6 + e_5 + f_5 + g_7 + g_6 + g_5 + h_7 + h_5 \\ e_6 + f_7 + f_6 + g_6 + h_7 + h_6 \\ e_7 + e_6 + e_5 + f_7 + f_5 + g_6 + g_5 + h_5 \end{cases}$$

Définition 3.2. *(Bits d'information) Dans la suite on désignera par* ***bits d'information*** *les bits nécessaires à l'inversion de MixColumns.*

La preuve provient simplement du développement du calcul de l'inverse de *MixColumns*. Elle est explicitée à l'annexe A.

Cette propriété implique que si on réalise une hypothèse sur 3 bits pour chacune des deux opérations *MixColumns* on pourra inverser en valeur cette opération. Ainsi, on peut inverser (en différence sur tout l'état et en valeur sur les quartets bas) un tour complet avec uniquement la donnée des quartets bas de l'état de sortie, des quartets bas de la clef et de 6 bits d'hypothèse.

Procédure naïve

Pour récapituler, les étapes d'inversion sur un tour d'une paire de messages (C, C') fonctionnent de la façon suivante. On suppose qu'on a réalisé une hypothèse sur les quartets bas de la clef-maître et que l'on possède les valeurs des quartets bas au point $(S, S')_t$ correspondant aux deux chiffrés (C, C')(voir figure 3.5) :

- On réalise une hypothèse sur les 6 bits indiqués dans la propriété 3.5 (3 bits pour chaque opération *MixColumns*).
- On obtient les valeurs des quartets à l'étape B, ce qui nous permet d'inverser *RotateNibbles*, *SubNibbles* et *AddRoundKey* sur les quartets bas. On obtient la moitié de la valeur de l'état à l'étape E pour les deux messages de notre paire, donc a fortiori la valeur de la différence à cet endroit.
- On inverse en différence l'opération linéaire *MixColumns* pour observer la différence à l'étape F.
- Si celle-ci se conforme au chemin différentiel, on conserve la paire et on recommence le processus pour le tour précédent. Sinon, on l'élimine.

On peut voir que cette façon de procéder requiert 2^6 inversions de tour (une pour chaque valeur possible des 6 bits d'information).

Optimisation

Il est possible de réduire cette complexité en temps de la façon suivante (on se réfère à la figure 3.6) :

On réalise dans un premier temps les hypothèses sur les 3 bits d'information permettant d'inverser l'opération *MixColumns* de droite ; cela nous permet de calculer la valeur des différences des quartets g', h', e et f et nécessite 2^3 inversions de tour.

D'après la propriété A.1 donnée en annexe A.2 (qui est l'analogue de la propriété 3.1 pour l'inverse de *MixColumns*), on sait que l'état F sera inactif sur les quartets hauts si et seulement si les quantités suivantes sont nulles :

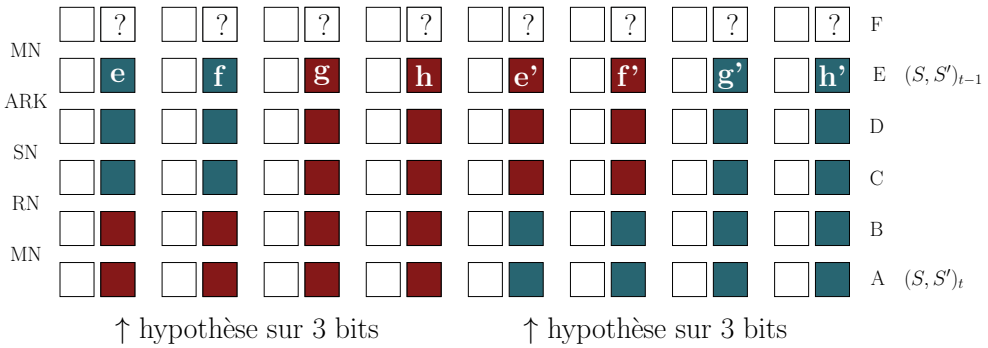


FIGURE 3.6 – Optimisation possible pour inverser un tour de KLEIN avec une complexité en temps de 2^4 déchiffrements de tour au lieu des 2^6 requis pour la version naïve.

$$\begin{cases} e_3 + f_3 + g_3 + h_3 = 0 \\ e_3 + e_2 + f_2 + g_3 + g_2 + h_2 = 0 \\ e_3 + e_2 + e_1 + f_3 + f_1 + g_2 + g_1 + h_1 = 0 \end{cases}$$

ainsi que :

$$\begin{cases} e'_3 + f'_3 + g'_3 + h'_3 = 0 \\ e'_3 + e'_2 + f'_2 + g'_3 + g'_2 + h'_2 = 0 \\ e'_3 + e'_2 + e'_1 + f'_3 + f'_1 + g'_2 + g'_1 + h'_1 = 0 \end{cases}$$

Pour optimiser les calculs on pré-calcule donc les quantités suivantes pour les 2^3 valeurs possibles des bits d'information :

$$\begin{cases} e_3 + f_3 \\ e_3 + e_2 + f_2 \\ e_3 + e_2 + e_1 + f_3 + f_1 \\ g'_3 + h'_3 \\ g'_3 + g'_2 + h'_2 \\ g'_2 + g'_1 + h'_1 \end{cases} \tag{3.1}$$

On réalise la même opération avec l'opération *MixColumns* de gauche. Les 3 bits d'information nous permettent de calculer g, h, e' et f', à partir desquels on calcule les 6 bits :

$$\begin{cases} g_3 + h_3 \\ g_3 + g_2 + h_2 \\ g_2 + g_1 + h_1 \\ e'_3 + f'_3 \\ e'_3 + e'_2 + f'_2 \\ e'_3 + e'_2 + e'_1 + f'_3 + f'_1 \end{cases} \tag{3.2}$$

Pour obtenir les ensembles de 6 bits d'information permettant d'obtenir une différence qui suit la différentielle on réalisera une fusion des 2 listes de 2^{-6} éléments, ce qui nous renverra en moyenne une seule valeur pour les quartets bas de la paire d'états $(S, S')_{t-1}$.

Pour résumer, cette procédure requiert $2^3 + 2^3 = 2^4$ inversions de tours et retourne en moyenne une seule valeur pour $(S, S')_{t-1}$: le nombre de candidats reste donc inchangé malgré l'hypothèse de 6 bits nécessaires.

Dans le cas où le chemin différentiel n'est pas itératif, les cribles que les paires doivent passer correspondent à la vérifications d'un plus grand nombre de conditions donc permettent d'éliminer plus de paires. Nous détaillerons ceci dans la section 3.3.1.

3.2.2 Optimisation de l'hypothèse sur les bits de clef-maître

Nous venons de voir que notre procédure d'inversion nécessitait de réaliser une hypothèse sur la moitié des bits de la clef-maître, soit 32, 40 ou 48 bits selon la version de KLEIN étudiée. Nous montrons ici qu'au lieu de réaliser cette hypothèse en une seule fois, il est possible de la fractionner de sorte à diminuer son coût et d'éliminer au plus vite une mauvaise hypothèse.

Pour cela, nous allons utiliser le premier tour du chemin différentiel, ce qui implique que la suite de l'attaque sera légèrement modifiée : au lieu d'inverser tous les tours les uns après les autres puis de comparer les valeurs obtenues avec celles du message clair, cette confrontation de valeurs sera faite à la fin du premier tour étant donné que notre amélioration implique de calculer le premier tour sur les quartets bas.

On commence par réaliser une hypothèse sur les 4 premiers quartets bas de la clef du premier tour [10], ce qui nous permet de calculer les opérations *AddRoundKey*, *SubNibbles* et *RotateNibbles*. On en déduit la différence en entrée de l'opération *MixColumns* du tour 1, ce qui nous permet de calculer la différence en sortie de cette opération et de s'assurer que celle-ci suit le chemin différentiel : si ce n'est pas le cas, on en déduit que l'hypothèse de 16 bits sur la clef est mauvaise. On construit une liste contenant l'ensemble des candidats possibles que l'on note L_1. Indépendamment, on exécute le même processus sur la seconde moitié de l'état : on réalise une hypothèse sur les 4 autres quartets bas de la clef du premier tour, que l'on teste grâce à la seconde opération *MixColumns* et on construit une seconde liste notée L_2. Le nombre de candidats de clef restant à la fin de cette étape dépend de la forme du chemin différentiel tronqué considéré : plus les conditions sur les différences sont strictes et moins de candidats seront conservés [11]. Aussi, certains chemins différentiels n'auront qu'un seul *MixColumns* actif, donc la liste associée contiendra 2^{16} éléments.

Les triplets considérés dans la suite de l'attaque sont formés par les couples (C, C') associés à une valeur candidate pour les quartets bas de la clef du tour 1, composée d'un élément de L_1 et un élément de L_2. Au total, on aura 2^{32+p_1} triplets candidats pour chaque paire de messages, où, rappelons-le, 2^{p_1} est la probabilité que le chemin différentiel soit suivi sur le premier tour.

La complexité en temps de cette étape est de $2^{16} + 2^{16} = 2^{17}$ chiffrements au lieu des 2^{32} d'une recherche naïve. Le nombre de triplets obtenus à la fin de cette étape vaut $2^{32+p_1} \times 2^{|K|-32}$ (le dernier facteur correspond aux hypothèses supplémentaires à réaliser avant de continuer l'attaque pour obtenir l'ensemble des quartets bas de la clef-maître K).

10. Par définition cette clef de tour correspond à la clef-maître tronquée à ses 64 premiers bits.
11. mais plus la probabilité du chemin différentiel sera petite !

3.2.3 Description générique et complexités

La première étape de l'attaque consiste à choisir une caractéristique différentielle tronquée recouvrant R tours (pour attaquer $R+1$ tours). On note sa probabilité 2^p, $p < 0$. Comme nous le verrons dans la section 3.3.1, il est possible d'obtenir divers compromis de complexité selon les différences choisies pour les 3 premiers tours (pour les autres tours on choisira toujours d'itérer la caractéristique de la figure 3.3).

Le premier paramètre important que définit le chemin différentiel est le nombre de bits de différence en entrée du chiffrement. Plus celui-ci sera important et plus l'attaquant pourra arranger ses messages clairs de façon judicieuse et réduire la quantité de messages nécessaires pour former les 2^{-p} paires requises. Une façon efficace d'organiser les messages est de composer ce que l'on appelle des *structures*. Cette technique, déjà utilisée par Eli Biham et Adi Shamir pour attaquer le DES [BS90], consiste à réutiliser un même message pour former plusieurs paires différentes. Pour un chemin différentiel ayant une différence d'entrée sur Δ_{in} bits, une structure sera formée des $2^{\Delta_{in}}$ messages parcourant l'ensemble des valeurs possibles sur ces Δ_{in} positions et dont la valeur des autres bits est fixée à une valeur quelconque mais identique pour tous les messages.

Propriété 3.6.
Si le nombre de bits de différence est Δ_{in}, chaque structure contiendra $2^{\Delta_{in}}$ messages ce qui permettra de construire environ $2^{2\Delta_{in}-1}$ paires de messages avec la différence souhaitée en entrée.

On notera par p_1, p_2 et p_3 les logarithmes des probabilités des tours 1, 2 et 3, respectivement. Les tours suivants étant couverts par la caractéristique itérative (de probabilité 2^{-6} par tour) on aura $p = p_1 + p_2 + p_3 + (-6) \times (R - 3)$.

L'attaque procède alors comme suit :

1. **Collecte des données** : En utilisant des structures, on demande la génération de suffisamment de chiffrés de sorte à obtenir les 2^{-p} paires nécessaires à l'attaque, ce qui correspond à chiffrer $\frac{2^{-p}}{2^{2\Delta_{in}-1}} 2^{\Delta_{in}}$ messages. En pratique on traitera une structure après l'autre ce qui nous permettra de n'avoir à stocker que $2^{\Delta_{in}}$ messages/chiffrés à la fois.
2. **Application du crible du dernier tour** : Comme cela a été fait dans l'attaque d'Aumasson et al. [ANS11], on élimine des paires qui, de façon sûre, ne suivent pas la caractéristique en inversant la dernière opération *MixNibbles* sur les différences des chiffrés. Toutes les paires pour lesquelles la différence obtenue n'est pas inactive sur les quartets hauts sont éliminées, ce qui correspond à un facteur de réduction de 2^{-32}. Pour réduire le nombre d'inversions nécessaires on pourra dans la pratique réaliser cette étape de la façon suivante : on inverse par *MixColumns* tous les chiffrés de la structure et on stocke dans une liste (triée) leurs quartets hauts. On y cherche ensuite les collisions. Au total il nous restera 2^{-p-32} paires après cette étape.
3. **Hypothèse initiale sur des bits de clef** : Pour chacune des 2^{-p-32} paires collisionnant à l'étape précédente, on réalise en deux étapes une hypothèse de 32 bits (comme expliqué à la section 3.2.2) correspondant aux 8 premiers quartets bas de la clef-maître. Comme la probabilité que chaque paire suive le chemin différentiel sur le premier tour est de 2^{p_1} on obtiendra $(2^{-p-32}) \times 2^{32} \times 2^{p_1}$ candidats à la fin de l'étape. Ceux-ci sont formés d'une paire de messages associée à 32 bits de clef. Si la version

attaquée est KLEIN-64, ces 32 bits correspondent à tous les quartets bas de la clef-maître. Au contraire, il faudra réaliser des hypothèses supplémentaires pour obtenir tous les quartets bas de la clef-maître des plus grandes versions : 8 bits pour KLEIN-80 et 16 bits pour KLEIN-96. Si on résume, à la fin de cette étape le nombre de triplets — formés d'une paire candidate et d'une valeur possible pour tous les quartets bas de la clef-maître — restants sera de : $2^{-p-32+32+p_1}$ pour KLEIN-64, $2^{-p-32+32+p_1+8}$ pour KLEIN-80 et $2^{-p-32+32+p_1+16}$ pour KLEIN-96. Cette étape nécessite un nombre d'opérations équivalent à $2 \times \frac{1}{12} \times 2^{16}$ chiffrements et permet de calculer les valeurs des quartets bas des paires candidates. On note $(S, S')_1^*$ cette paire.

4. **Inversion des tours** : Dans cette étape on reconsidère la fin du chiffrement et on part des chiffrés des triplets restants pour inverser les tours les uns après les autres, comme expliqué à la section 3.2.1. Chaque inversion de tour nécessite une complexité en temps correspondant à 2^4 évaluations d'un tour pour chaque triplet. Lors de l'inversion des tours couverts par le chemin différentiel de la figure 3.3, les coûts des hypothèses faites pour inverser l'opération *MixNibbles* sont exactement compensés par le facteur de réduction de la vérification du chemin différentiel, ce qui implique que le nombre de triplets candidats reste constant. Au contraire, les premiers tours du chiffrements permettent d'obtenir des réductions plus importantes et de ce fait de réduire le nombre de candidats. Ces étapes sont réalisées pour les différents triplets les uns après les autres. Si un triplet n'est pas éliminé par les cribles (car il suit le chemin différentiel) on aura obtenu à la fin de l'inversion la valeur des quartets bas de $(S, S')_1$. On réalisera 6 hypothèses supplémentaires pour inverser le premier *MixNibbles* ce qui nous permettra de confronter les valeurs des quartets bas ainsi que les différences obtenues avec ce qui a été précédemment calculé pour $(S, S')_1^*$. Si on note 2^q le facteur de réduction correspondant à l'inversion des tours 2, 3 et 4, le nombre total de candidats après avoir inversé ces étapes sera de $2^{-p-32+32+p_1+6\times4+q}$ dans le cas de KLEIN-64, $2^{-p-32+32+p_1+8+6\times4+q}$ dans le cas de KLEIN-80, et $2^{-p-32+32+p_1+16+6\times4+q}$ dans le cas de KLEIN-96.

5. **Recouvrement de clef** : Pour obtenir la clef complète nous réalisons une recherche exhaustive sur les quartets hauts de la clef. Cette étape est de complexité négligeable comparée aux précédentes.

3.3 Résultats

3.3.1 Résultats pour plusieurs compromis possibles

Il est possible de choisir différents chemins dans les premiers tours du chemin différentiel, ce qui a une influence sur plusieurs facteurs parmi lesquels :

— la probabilité du chemin différentiel, donc sur la quantité de paires clairs/chiffrés nécessaires pour mener l'attaque,
— la taille des structures, donc la complexité en mémoire,
— l'efficacité des cribles, donc le nombre de candidats restant à la fin des étapes d'inversion de tour.

Les quatre débuts de chemins différentiels les plus intéressants parmi ceux que nous avons étudiés sont représentés à la figure 3.7.

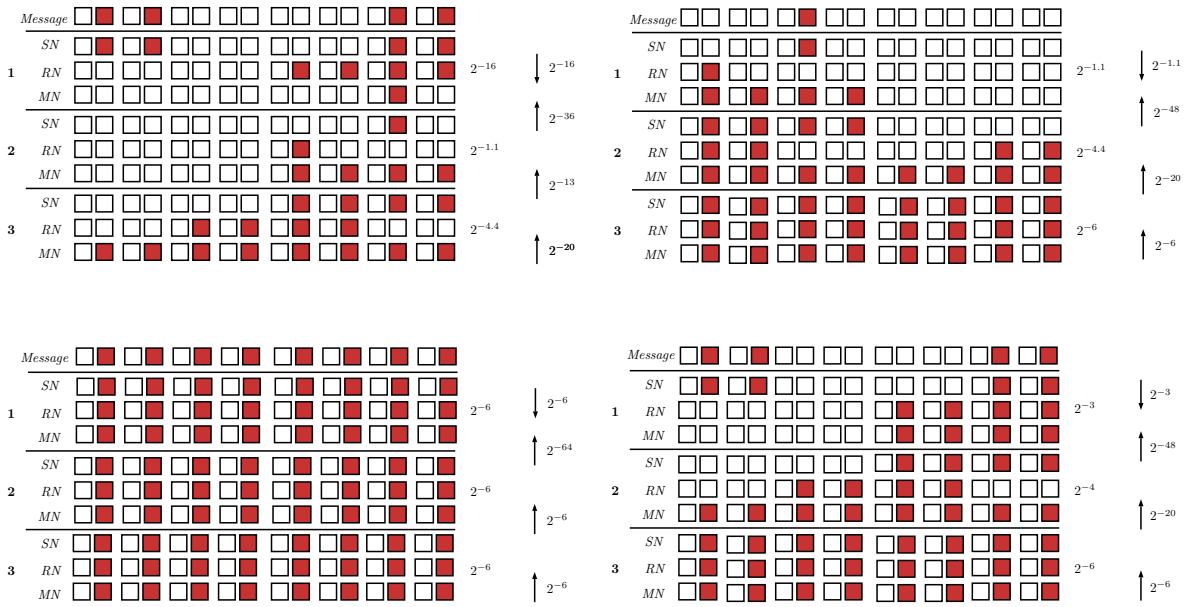


FIGURE 3.7 – Chemins différentiels possibles pour obtenir différents compromis (de gauche à droite et de bas en haut : cas 1, 2, 3 et 4).

Cas	Données	Temps	Mémoire
1	$2^{54.5}$	2^{57}	2^{16}
2	$2^{56.5}$	2^{62}	2^{4}
3	2^{35}	$2^{63.8}$	2^{32}
4	2^{46}	2^{62}	2^{16}

TABLE 3.3 – Complexités de l'attaque sur la version complète de KLEIN-64 avec les 4 compromis étudiés.

Le **cas 1** correspond au chemin différentiel tronqué représenté à la figure 3.8 les tours 4 à 11 sont couverts par la caractéristique itérative, alors que la caractéristique des premiers tours est la suivante :

tour 1 : passage d'une différence dans les quartets 1, 3, 13 et 15 à une différence sur l'unique quartet 13. La différence en entrée du tour 1 porte sur 4 quartets bas, positionnés de sorte à ce qu'ils entrent tous dans le même *MixColumns* (celui de droite sur la figure 3.8), la caractéristique visée possède une seule boîte-S active après le passage de cette étape. Si on se réfère aux équations induites par ce passage de *MixColumns*, on peut compter qu'il correspond à la vérification de 16 équations linéaires, donc il est satisfait avec probabilité 2^{-16}.

tour 2 : passage d'une différence portant sur le quartet 13 à une différence portant sur les quartets 9, 11, 13 et 15. La différence d'entrée de ce tour porte sur un seul quartet, qui a donc une différence strictement non nulle. Cette différence est modifiée par passage par la boîte-S, puis le quartet actif est déplacé en position 9. Finalement, il rentre dans l'opération *MixColumns* qui, étant donné sa propriété *MDS*, donnera lieu à 4 octets actifs.

La probabilité que la différence en sortie de cette opération porte uniquement sur les quartets bas correspond à la probabilité que le bit de poids fort du quartet actif vaille 0. Il y a exactement 7 cas favorables (et non pas 8 car la différence ne doit pas être entièrement nulle) sur un total de 15 possibilités, donc une probabilité de $2^{-1.1}$.

tour 3 : passage d'une différence portant sur les quartets 9, 11, 13 et 15 (tous les 4 non nuls) à une différence portant uniquement sur des quartets bas. Après le passage de l'opération *SubNibbles*, les quartets actifs sont déplacés par *RotateNibbles* : 2 quartets actifs rentrent dans le premier *MixColumns*, les 2 autres rentrent dans le second. La différence ne se propagera pas dans les quartets hauts sous la condition que les 2 bits de poids fort des 2 quartets actifs de l'entrée sont tous les deux nuls : pour chaque opération *MixColumns* on a 49 événements favorables sur les 225 possibles, donc ce tour est passé correctement avec probabilité $(2^{-2.2})^2 = 2^{-4.4}$.

La probabilité de la caractéristique sur 11 tours vaut $2^{-69.5}$. Comme la différence en entrée couvre 4 quartets, on peut utiliser les structures et limiter le nombre de couples clairs-chiffrés nécessaires à $2^{54.5}$. La complexité en temps de l'attaque est de 2^{57}.

Les cas 2, 3 et 4 sont construits par composition des mêmes caractéristiques sur un tour.

Le **cas 2** reprend le chemin différentiel utilisé par Aumasson et al. [ANS11], mais au lieu de fixer l'unique quartet actif à la valeur `0xb`, il considère juste que le quartet est actif. En conséquence la taille des structures utilisées est de 2^4 donc la mémoire nécessaire sera très petite. La probabilité du chemin obtenu pour attaquer KLEIN-64 sera de $2^{-1.1-4.4-6\times 9} = 2^{-59.5}$ donc l'attaque nécessitera $2^{59.5-7+4} = 2^{56.5}$ couples clairs-chiffrés. La complexité en temps de l'attaque est dominée par le terme : $2^{59.5-32+32-1.1} \times 2 \times \frac{1}{2} \times \frac{9}{12} \times (2^3+2^3) \simeq 2^{61.98}$. À la fin de la procédure d'inversion il reste environ $2^{7.4}$ candidats, menant à la clef avec un coût négligeable.

Le **cas 3** correspond à un chemin entièrement itératif. En conséquence, les structures contiennent 2^{32} éléments (ce qui mène à l'attaque de plus faible complexité en données : 2^{35}, mais en contrepartie à la pire complexité en mémoire) et la probabilité du chemin est de $2^{-6\times 11} = 2^{-66}$.

Le **cas 4** utilise un chemin différentiel permettant de s'assurer qu'une seule opération *MixColumns* est active au premier tour, donc considère une différence d'entrée avec les 2 premiers et les 2 derniers quartets bas actifs. Le chemin sur 11 tours a une probabilité de $2^{-3-4-6\times 9} = 2^{-61}$ et l'attaque nécessite $2^{16+30} = 2^{46}$ couples clairs-chiffrés.

Le calcul des probabilités que les opérations *MixNibbles* des tours 2 et 3 soient inversées de façon conforme au chemin différentiel se fait encore une fois en se reportant aux équations explicitées en annexe A.2. Par exemple, inverser l'opération *MixNibbles* au tour 3 du cas 1 de façon conforme revient à vérifier 10 équations pour chacun des *MixColumns*, donc a un facteur de réduction de 2^{-20}. Les facteurs de réduction sont explicités sur la figure 3.7.

Les réductions obtenues en inversant l'opération *MixNibbles* du tour 1 sont beaucoup plus efficaces puisqu'elles correspondent à la vérification de la coïncidence des valeurs des différences et des valeurs des états à ce point. Pour le cas 1, on vérifie la concordance (en valeur) de 8 quartets (2^{-32}) et de 1 quartet de différence (2^{-4}) donc on obtient un facteur de réduction de 2^{-36}. Les cas 2 et 4 correspondent à la vérification de 32 bits en valeurs et 16

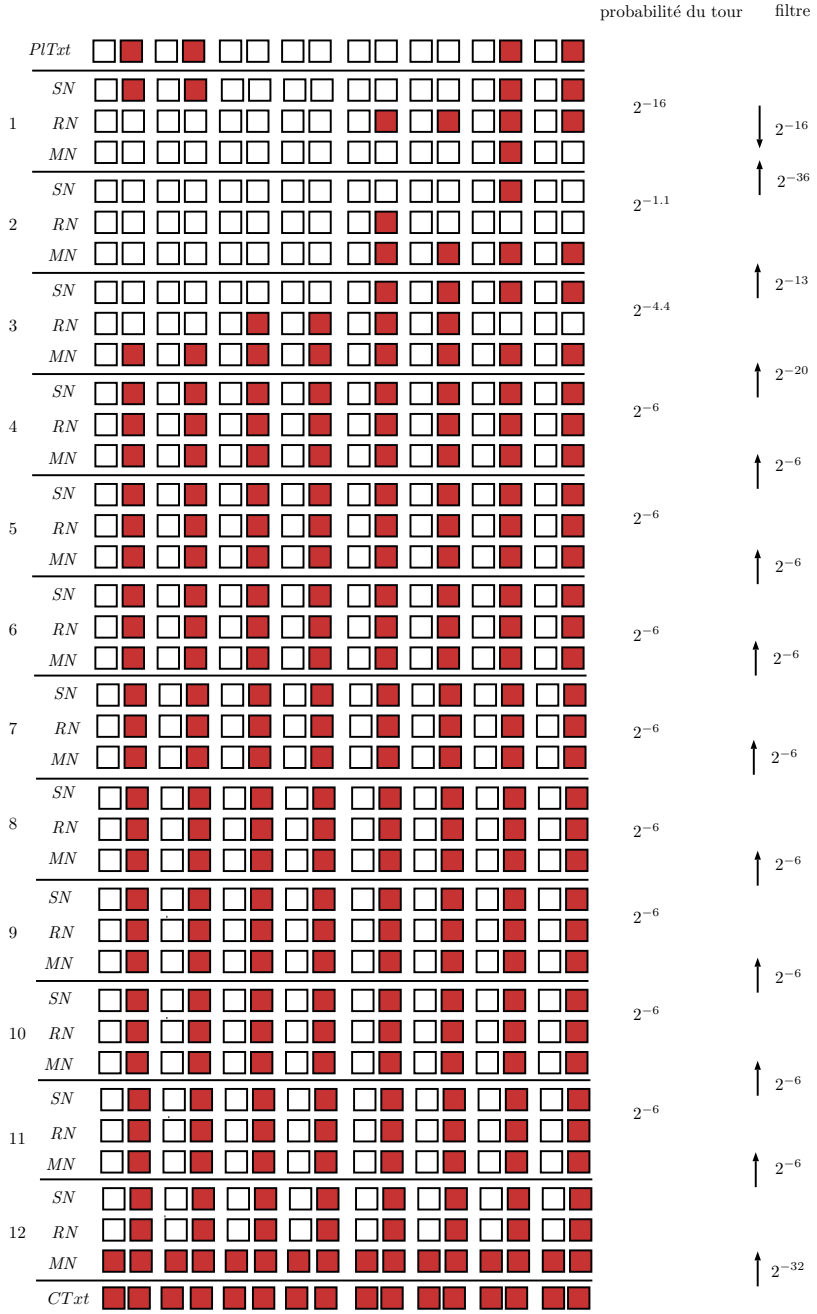


FIGURE 3.8 – Attaque sur la version complète de KLEIN-64 utilisant le cas 1

en différences donc à un crible de probabilité 2^{-48} et finalement comme le cas 3 vérifie une concordance sur 32 bits en valeurs et en différence on a un facteur de réduction de 2^{-64}.

3.3.2 Attaques sur KLEIN-80 et KLEIN-96

En appliquant nos attaques aux versions de KLEIN possédant des clefs plus grandes nous obtenons les résultats consignés à la table 3.4. Pour qu'une attaque soit valide pour ces versions il faudra qu'elle ait une complexité en données inférieure à ce que contient le *codebook* soit 2^{64} messages. Les complexités en temps doivent être inférieures à 2^{80} pour KLEIN-80 et à 2^{96} pour KLEIN-96. Le paramètre devenant le plus rapidement limitant est sans aucun doute la complexité en données, et c'est pourquoi les attaques réussissant à couvrir le plus de tours sont celles tirant parti de grandes structures (donc les cas 3 et 4). Notre attaque permet d'attaquer 13 tours sur les 16 que compte KLEIN-80 et 14 tours sur les 20 que compte KLEIN-96.

Version	Cas	Tours	Données	Temps	Mémoire
80	1	12	$2^{54.5}$	2^{65}	2^{16}
80	1	13	$2^{60.5}$	$2^{71.1}$	2^{16}
80	2	13	$2^{62.5}$	2^{76}	2^{4}
80	3	13	2^{41}	2^{78}	2^{32}
80	4	13	2^{52}	2^{76}	2^{16}
96	3	14	2^{47}	2^{92}	2^{32}
96	4	14	$2^{58.4}$	$2^{89.2}$	2^{16}

TABLE 3.4 – Résultats de nos attaques sur des versions réduites de KLEIN-80 et KLEIN-96.

3.3.3 Vérification expérimentale

Pour confirmer la validité de notre attaque et de nos estimations de complexité, nous avons implémenté l'attaque utilisant le chemin différentiel tronqué du *cas 1* sur 9 et 10 tours. Nous retrouvons les quartets bas de la clef grâce au processus d'inversion des tours et les quartets hauts sont déterminés ensuite par une recherche exhaustive un peu améliorée.

Pour réaliser nos séries d'expérimentations nous avons utilisé un ordinateur de bureau avec un processeur Intel(R) Xeon(R) CPU W3670 à 3.20 GHz (12 MB cache) et 8 GB de RAM. Les performances auraient sans conteste pu être améliorées en parallélisant le programme, par exemple en traitant 1 structure par processus.

Chacune des exécutions a retrouvé la bonne clef.

Simulation sur 9 tours

L'attaque considérée utilise les 8 premiers tours du chemin différentiel représenté à la figure 3.8, le tour 9 étant laissé libre. La probabilité du chemin vaut donc $2^{-16-1.1-4.4-6\times5} = 2^{-51.5}$. Comme pour ce chemin on a $\Delta_{in} = 16$, une structure contiendra 2^{31} paires de messages et on s'attend donc à avoir besoin de $2^{20.5}$ structures pour monter l'attaque.

Nous reportons ci-dessous le résultat retourné par une des simulations sur 9 tours :

```
NB rounds: 9
MasterKeyToFind:         b4 83 d4 f2  2a 61 20 b8

structure:               30 70 da 10  8c 53 f0 00
Plaintext 1:             3e 7e da 10  8c 53 f4 0d
Plaintext 2:             3c 73 da 10  8c 53 f8 09
lower nibbles found:     04 03 04 02  0a 01 00 08
Complete Key:            b4 83 d4 f2  2a 61 20 b8
number of pltxt:         70445946095  ie: 2^36.035798
number of false alarms:  536892       ie: 2^19.034272
number of structures:    1074920
time elapsed:            123340.140000 sec
```

La première ligne indique la clef-maître utilisée pour générer l'ensemble des paires mises à disposition de l'attaquant. Les 3 lignes suivantes (`structure`, `Plaintext 1` et `Plaintext 2`) font référence à la bonne paire trouvée lors de l'attaque : celle-ci faisait partie de la structure formée par tous les messages clairs dont tous les quartets étaient fixés hormis les 2 premiers et les 2 derniers quartets bas (ce sont les 4 quartets colorés sur la figure 3.8). On avait donc la structure formée de tous les messages de 64 bits de la forme

`3* 7* da 10  8c 53 f* 0*`

dont les 4 quartets marqués d'une étoile parcouraient toute les possibilités (soit au total 2^{16} messages). On peut vérifier que les quartets bas de clef trouvés dans la première phase de l'algorithme sont corrects. Ceux-ci sont complétés dans la seconde phase de l'attaque pour former la clef complète correcte. Le nombre de messages clairs qui ont été chiffrés lors de l'attaque est d'environ 2^{36}, et la simulation indique que $1074920 \approx 2^{20}$ structures différentes ont été nécessaires (ces chiffres sont cohérents puisqu'on a vu que chaque structure permet de générer 2^{16} messages). Le champ `number of false alarms` fait référence au nombre de paires passant le premier test (correspondant à l'inversion de *MixNibbles* sur la différence des chiffrés pour s'assurer que seuls les quartets bas sont actifs). Ici on a environ 2^{19} telles paires. Là encore, cela concorde avec la théorie : les 2^{20} structures nous permettent de former environ $2^{20} \times 2^{16+15} = 2^{51}$ paires. Sachant que la probabilité de passer le premier test est de 2^{-32}, on s'attend à ce que 2^{19} paires passent. Toutes (sauf une) sont des fausses alarmes dans le sens où bien qu'elles passent ce crible elles ne suivent pas le chemin différentiel.

Cette simulation a duré un peu plus de 34 heures. Nous avons vérifié manuellement que la paire retournée suivait la caractéristique tronquée et c'est bien le cas.

Nous avons lancé 4 simulations au total ce qui nous a permis d'obtenir les moyennes suivantes :

— $2^{36.48}$ messages chiffrés
— $2^{19.48}$ fausses alarmes,

ce qui est très proche de ce que nous avions estimé en théorie ($C_D = 2^{36.5}$ chiffrés et $2^{51.5-32} = 2^{19.5}$ fausses alarmes).

Simulation sur 10 tours

Nous avons ensuite adapté le programme pour qu'il teste la version de l'attaque sur 10 tours, et nous avons réussi à obtenir la première attaque pratique sur autant de tours (nécessitant tout de même près de 15 jours de calcul).

Le chemin différentiel considéré consiste en les 9 premiers tours de celui représenté à la figure 3.8, et a donc une probabilité de $2^{-57.5}$ (on rajoute 1 tour de probabilité 2^{-6} par rapport au cas précédent).

Les résultats de l'exécution sont reportés ci-dessous. On peut y voir que, par chance, elle a nécessité un peu moins de données que prévu ($2^{42.5}$) mais que le ratio de paires passant le test du dernier tour est bien de 2^{-32} : nous avons chiffré $2^{-39.18}$ messages ce qui permet de former $2^{54.18}$ paires de messages dont $2^{54.18-32} = 2^{22.18}$ sont supposées passer le premier test.

```
NB rounds: 10
MasterKeyToFind:          66 a2 fa 17  23 19 39 bd

structure:                c0 70 03 39  de 72 30 e0
Plaintext 1:              c3 79 03 39  de 72 34 e4
Plaintext 2:              c2 77 03 39  de 72 30 e1
lower nibbles found:      06 02 0a 07  03 09 09 0d
Complete Key:             66 a2 fa 17  23 19 39 bd
number of pltxt:          624712959124  ie: 2^39.184403
number of false alarms:   4764629       ie: 2^22.183932
number of structures:     9532364
time elapsed:             1254407.310000 sec
```

3.4 Conclusion

Dans ce chapitre nous avons montré que le chiffrement KLEIN-64 n'apportait pas la sécurité annoncée par ses auteurs.

Le principal problème que nous avons mis en évidence et exploité dans notre analyse est la trop faible diffusion — au niveau des quartets — apportée par l'opération *MixColumns*. Comme annoncé au début de ce chapitre, le choix de l'opération *MixColumns* a été justifié par les auteurs par des arguments liés à la performance du système : étant donné que l'opération *MixColumns* est définie par une multiplication matricielle orientée octets, elle permet d'obtenir une implémentation logicielle plus efficace sur les processeurs 8 bits. Bien que les concepteurs aient eu conscience qu'une étape de diffusion orientée quartets aurait été préférable, l'argument de performance les a fait opter pour cette solution qui, malheureusement, s'est avérée néfaste.

Incontestablement, la première chose à modifier dans ce chiffrement pour qu'il résiste à nos attaques (et qui serait aussi efficace contre les attaques de type PC-MITM) serait de changer la fonction de diffusion pour apporter une meilleure diffusion au niveau des quartets. En plus de changer la définition de l'opération *MixColumns* en optant pour une autre matrice que celle de *MixColumns*, il serait judicieux d'apporter plus de diffusion dans l'algorithme de cadencement de clef.

Chapitre 4

Cryptanalyse de réseaux de substitution-permutation avec couche non-linéaire partielle

Ce chapitre détaille un travail réalisé avec Achiya Bar-On, Itai Dinur, Orr Dunkelman, Nathan Keller et Boaz Tsaban et présenté à Eurocrypt 2015 [BDD+15]. Les recherches menées ici portent sur la construction proposée par Benoît Gérard et ses coauteurs à CHES 2013 [GGNS13] dont l'objectif était d'offrir un chiffrement par blocs à bas coût qui soit facile à protéger contre les attaques par canaux auxiliaires. Leur construction est un réseau de substitution-permutation pour lequel seule une partie de l'état est modifiée par l'opération non-linéaire de la fonction de tour. Plus que la cryptanalyse de l'instance concrète proposée dans [GGNS13] (appelée `Zorro`), nos recherches ont permis d'établir des techniques génériques d'analyse de ce type de constructions sous l'angle de leur résistance aux attaques différentielles et linéaires. Une des conclusions les plus importantes de nos travaux est sans doute que ce type de construction n'est pas mauvais en soi et qu'il est possible de trouver des instances évitant les faiblesses détectées dans `Zorro`.

4.1 Problématique des attaques par canaux auxiliaires

4.1.1 Introduction

Jusqu'à la fin du 20ème siècle[1] et l'apparition des travaux fondateurs de Paul Kocher [Koc96], la sécurité d'un algorithme était uniquement évaluée dans le modèle en boîte noire, c'est-à-dire qu'un attaquant était supposé ne pouvoir avoir accès qu'à l'entrée et à la sortie d'un algorithme cryptographique (voir figure 4.1). Ce modèle fut amélioré en celui représenté à la figure 4.2 pour rendre compte de la réalité *physique* de l'utilisation d'un algorithme. Il considère que l'attaquant a accès à l'appareil exécutant l'algorithme et peut mesurer ses émissions (les fuites de ses *canaux auxiliaires*) et ainsi obtenir de l'information sur son fonctionnement et sur les données secrètes manipulées. Ce type d'attaques est lié à l'implémentation effective de l'algorithme et est indépendant de la robustesse cryptographique de celui-ci : un algorithme parfaitement invulnérable aux attaques dites *logiques* pourra se

1. Du moins en ce qui concerne le domaine public.

montrer désastreux s'il est mal implémenté.

Plusieurs paramètres peuvent donner de l'information sur l'algorithme et sur les données en cours de traitement, comme par exemple la consommation de courant, les radiations électro-magnétiques émises, le temps d'exécution ou encore le son généré lors des calculs.

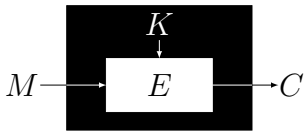


FIGURE 4.1 – Modèle en boîte noire : l'attaquant a uniquement accès au message clair M, au chiffré C et connaît l'algorithme E.

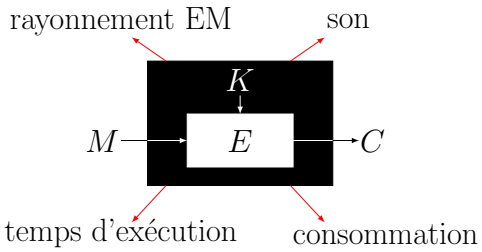


FIGURE 4.2 – Situation réelle : l'attaquant a accès au dispositif physique exécutant l'algorithme cryptographique, donc à tout ce que celui-ci laisse fuir : ondes électromagnétiques, son, etc.

On distingue deux types d'attaques utilisant les fuites physiques d'un dispositif, selon que l'attaquant se permette de perturber le système ou non : s'il se contente d'effectuer des mesures on parlera *d'attaques par canaux auxiliaires*[2] sinon ce sera une attaque dite *par fautes*[3].

Un des résultats récents les plus impressionnants d'attaque par canaux auxiliaires est sans doute l'attaque acoustique menée par Daniel Genkin et ses coauteurs : dans leur travail intitulé *RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis* [GST14], ils montrent comment analyser le son généré par un ordinateur portable pour en extraire la clef RSA de 4096 bits utilisée dans GnuPG, et cela en moins d'une heure.

4.1.2 Techniques de protection contre les attaques par canaux auxiliaires

Rapidement après la prise de conscience de cette menace, tout un ensemble de contre-mesures furent proposées, comme le *blindage* pour limiter le rayonnement électromagnétique ou l'ajout d'opérations inutiles pour lisser la consommation électrique. De façon générale il existe 2 catégories de contre-mesures : celles visant à empêcher au maximum les fuites physiques et celles décorrélant les fuites physiques des données secrètes.

2. En anglais *Side Channel Attacks - SCA*.
3. En anglais *Fault attack*.

La contre-mesure la plus populaire entrant dans cette dernière catégorie est une technique dérivée du *partage de secret*[4]. L'objectif de celle-ci est de partager une information secrète entre plusieurs entités de sorte que le secret ne puisse être retrouvé que si un nombre préalablement défini d'entités mettent en commun leurs données.

Définition 4.1. *(Schéma de partage de secret de Shamir [Sha79]) Le schéma de partage de Shamir utilise le principe de l'interpolation polynomiale : pour caractériser un polynôme de degré d il suffit de posséder $d+1$ évaluations différentes de celui-ci, alors que la connaissance de strictement moins de $d+1$ points n'apporte aucune information. On peut alors convenir que le secret correspond à l'évaluation du polynôme en $x = 0$ et donner à chaque entité l'évaluation du polynôme en un point quelconque non nul. Si on souhaite qu'un minimum de s parts soient nécessaires pour retrouver le secret (s est appelé le seuil) on utilisera un polynôme de degré $s+1$.*

Dans le contexte des attaques par canaux auxiliaires, les techniques de partage de secret furent rebaptisées *schémas de masquage*. Les premiers travaux proposant d'utiliser les techniques de partage de secret dans ce contexte furent réalisés en 1999 dans [GP99] et [CJRR99]. Leur mise en œuvre permet de protéger l'implémentation d'un chiffrement par blocs des attaques du type de celles basées sur la consommation de courant et sur les radiations électromagnétiques en éliminant la corrélation directe entre les données sensibles et les fuites physiques.

L'idée de base est de partager chaque donnée sensible en $d+1$ parts[5] ou *shares* de la même manière que lors d'un partage de secret : la combinaison de toutes les parts doit permettre de retrouver la donnée initiale mais une part seule ou jusqu'à d parts prises ensemble doivent être indépendantes de la donnée sensible. Cette technique apparaît clairement comme une réponse efficace aux attaques par canaux auxiliaires lorsqu'on considère le *Probing Model*. Ce modèle estime que l'attaquant a la possibilité d'obtenir les fuites physiques correspondant aux valeurs circulant sur un nombre limité de fils d'un circuit booléen. L'algorithme cryptographique implémenté réalisera alors uniquement des opérations sur ces parts de sorte que les fuites physiques qu'il produit ne soient pas directement liées aux données sensibles.

De façon assez naturelle il est tout de même possible de réaliser une attaque sur une telle implémentation puisqu'il suffit de cibler $d+1$ instants différents correspondant à la manipulation des différentes parts. C'est ce qu'on appelle une attaque par *canaux auxiliaires d'ordre supérieure* (*Higher-Order-Side-Channel-Attack*). Nous n'allons pas rentrer dans les détails de ces attaques ici mais le point important est que leur difficulté croît exponentiellement en l'ordre du masquage pour des raisons physiques provenant du bruit contenu dans les observations physiques.

La littérature utilise la dénomination de *schéma de masquage* pour désigner toutes les contre-mesures des attaques par canaux auxiliaires utilisant le principe de décomposition des données sensibles en parts. Malgré cette dénomination commune il existe une multitude de schémas distincts qui diffèrent notamment dans la façon de modifier l'algorithme cryptographique de sorte que les calculs puissent être faits de façon indépendante sur chacune des parts.

Dans la pratique, si on souhaite protéger à l'ordre d la donnée sensible X on générera au début de chaque chiffrement d nombres aléatoires M_1 à M_d puis on calculera M_0 de sorte que

4. Ou *secret sharing*.
5. On appelle d l'ordre du schéma de masquage.

la relation suivante soit satisfaite :

$$M_0 \star M_1 \star M_2 \star \cdots \star M_d = X$$

Selon le type de chiffrement que l'on souhaite protéger, on choisira $\star$ comme étant l'addition modulaire (on parlera alors de *masquage arithmétique*) ou comme étant le `XOR` binaire (on parlera alors de *masquage booléen*), le schéma multiplicatif correspondant à choisir la multiplication est une autre possibilité. La difficulté de cette technique consiste à trouver quelles opérations réaliser sur les parts de sorte que, lorsqu'on recombine les résultats finaux, on obtienne le chiffré.

Pour illustrer notre propos, considérons que l'on souhaite protéger à l'ordre d un vecteur booléen sur k bits $X \in \mathbb{F}_2^k$. La première étape consiste à générer d nombres aléatoires M_1 à M_d et à calculer M_0 de sorte que :

$$M_0 \oplus M_1 \oplus \cdots \oplus M_d = X$$

Si on souhaite calculer $L(X)$ où L est une fonction linéaire de $\mathbb{F}_2^k$ dans lui-même, alors effectuer cette opération de façon masquée est très simple, puisque par linéarité on a :

$$L(M_0) \oplus L(M_1) \oplus \cdots \oplus L(M_d) = L(X)$$

ce qui implique que calculer L sur la donnée secrète peut se faire en calculant L sur chacune des parts.

Une fonction affine sera aussi très facile à gérer : on pourra par exemple appliquer la transformation affine à chaque part puis, si le nombre de parts est pair, ajouter la partie constante à une seule des parts.

Les élévations à une puissance dont l'exposant est une puissance de 2 sont elles aussi directes, puisque en caractéristique 2 c'est une opération linéaire :

$$M_0^{2^i} \oplus M_1^{2^i} \oplus \cdots \oplus M_d^{2^i} = X^{2^i}$$

La difficulté provient du calcul des opérations non-linéaires S puisque, bien entendu, pour celles-ci on aura :

$$S(M_0) \oplus S(M_1) \oplus \cdots \oplus S(M_d) \neq S(X)$$

4.1.3 Le schéma de masquage de Rivain et Prouff

En 2003, Ishai, Sahai et Wagner [ISW03] proposèrent une technique permettant de masquer à tout ordre les portes AND et NOT d'un circuit booléen. Ces deux portes étant suffisantes pour décrire tout circuit, leur technique permet de masquer un circuit booléen quelconque. À CHES 2010, Rivain et Prouff [RP10] étendirent ces techniques afin qu'elles s'appliquent aux multiplications dans un corps fini. Leur solution est décrite dans l'algorithme 2.

Données : parts M_i correspondant à X ($\bigoplus_{i=0}^{d} M_i = X$) et parts N_i correspondant à Y ($\bigoplus_{i=0}^{d} N_i = Y$)
Résultat : parts P_i vérifiant $\bigoplus_{i=0}^{d} P_i = XY$ sur $\mathbb{F}_2^n$
pour i *de* 0 *à* d **faire**
  pour j *de* $i+1$ *à* d **faire**
    $r_{i,j} \leftarrow$ nombre aléatoire sur n bits
    $r_{j,i} \leftarrow (r_{i,j} \oplus M_i N_j) \oplus M_j N_i$
  fin
fin
pour i *de* 0 *à* d **faire**
  $P_i \leftarrow M_i N_i$
  pour j *de* 0 *à* d, $j \neq i$ **faire**
    $P_i \leftarrow P_i \oplus r_{i,j}$
  fin
fin

Algorithme 2 : Masquage à l'ordre d de la multiplication sur un corps fini $\mathbb{F}_2^n$ [RP10].

Il est facile de voir que cette multiplication sécurisée à l'ordre d demande :
— $(d+1)^2$ multiplications dans $\mathbb{F}_2^n$,
— $2d(d+1)$ `XORs`,
— $d(d+1)/2$ nombres aléatoires de n bits.

Cet algorithme leur permit de proposer un schéma de masquage de l'AES, et plus particulièrement de gérer le masquage à l'ordre d de sa boîte-S. Rappelons ici que celle-ci est donnée par une transformation affine sur $\mathbb{F}_2$ de la sortie de l'application :

$$\begin{array}{ccc} \mathbb{F}_2^8 & \to & \mathbb{F}_2^8 \\ x & \mapsto & x^{254} \end{array}$$

L'idée de Rivain et Prouff est de réaliser cette exponentiation en l'exprimant comme une succession d'élévations à une puissance 2^i et de multiplications, en minimisant autant que possible ces dernières. La décomposition qu'ils exploitent fait intervenir 4 multiplications, ce qui est le minimum possible :

$$x \xrightarrow{.^2} x^2 \xrightarrow{\times x} x^3 \xrightarrow{.^4} x^{12} \xrightarrow{\times x^3} x^{15} \xrightarrow{.^{16}} x^{240} \xrightarrow{\times x^{12}} x^{252} \xrightarrow{\times x^2} x^{254}$$

Cette construction permet de masquer la boîte-S de l'AES avec :
— 4 multiplications sécurisées dans $\mathbb{F}_2^8$,
— 1+2+4 = 7 élévations au carrés,
— 1 multiplication par une matrice (correspondant à l'application affine appliquée à l'élévation à la puissance 254).

4.2 Les réseaux de substitution-permutation avec couche non linéaire partielle et le chiffrement Zorro

Au vu de la section précédente, le surcoût induit par le masquage d'un chiffrement par blocs est dominé par le coût de masquage des opérations non-linéaires : si on choisit d'utiliser

le schéma proposé par Rivain et Prouff à CHES 2010, le paramètre essentiel à minimiser est le nombre total de multiplications dans le corps fini réalisées par le chiffrement.

Deux voies de recherche naturelles s'offrent alors à un concepteur souhaitant construire un chiffrement qui reste performant après masquage :

— La première consiste à utiliser des boîtes-S dont la description comporte peu d'opérations non-linéaires (et si possible dans un petit corps fini) : cette stratégie fut celle adoptée par Piret, Roche et Carlet pour concevoir le chiffrement PICARO [PRC12a] que nous étudierons au chapitre 5.

— La seconde consiste à opter pour un algorithme de chiffrement dont la description possède peu d'appels aux boîtes-S.

Le chiffrement que nous étudions à présent, dénommé `Zorro` (en référence au justicier masqué de Johnston McCulley) suit ces deux stratégies : sa fonction de tour réalise uniquement 4 applications de boîtes-S (et au total le chiffrement complet en compte environ 2 fois moins que l'AES-128) et celles-ci nécessitent un petit nombre de multiplications sécurisées (une de moins que celle de l'AES, et dans un corps fini plus petit).

On peut voir ce type de constructions comme un compromis entre les deux constructions classiques utilisées pour les chiffrements par blocs que sont :

— Les réseaux de substitution-permutation pour lesquels les opérations linéaires et non-linéaires couvrent la totalité de l'état à chaque tour

— Les réseaux de Feistel pour lesquels ces opérations ne couvrent qu'une partie de l'état interne (la moitié dans le cas des réseaux de Feistel de base, mais possiblement d'autres rapports dans le cas des réseaux de Feistel généralisés).

De façon générale dans toute la suite on désignera par le nom de *Partial Substitution Permutation Networks* (que l'on abrégera en **PSPN**) les chiffrements SPN pour lesquels l'opération non-linéaire ne porte que sur une partie de l'état interne.

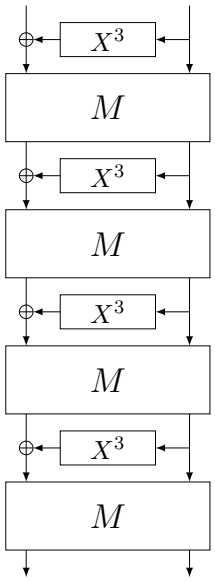
4.2.1 Description du chiffrement `Zorro`

L'approche suivie par les concepteurs de `Zorro` consiste à partir de l'AES et à étudier comment modifier sa construction de sorte que le chiffrement obtenu soit à la fois sûr et optimal pour les deux métriques liées au schéma de masquage (le nombre d'opérations non linéaires et la quantité de nombres aléatoires à générer).

Boîte-S

La première étape de leur travail fut la recherche d'une boîte-S bijective de 8 bits plus performante que celle de l'AES. Comme une recherche exhaustive parmi les $2^8!$ permutations possibles n'était bien entendu pas envisageable, ils limitèrent leur recherche à deux types de boîtes-S : celles pouvant s'exprimer comme un polynôme à 1 ou 2 monômes et celles pouvant se construire à partir de boîtes-S plus petites. Leur analyse les conduisit à préférer cette seconde option et à opter pour une boîte-S construite par 4 tours d'un schéma de Feistel comme représenté figure 4.3. La fonction de tour de ce schéma de Feistel correspond à l'application :

$$\begin{array}{rcl} f : \mathbb{F}_2^4 & \to & \mathbb{F}_2^4 \\ X & \mapsto & X^3. \end{array}$$



où l'application linéaire M est définie de la façon suivante :

$$M = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

FIGURE 4.3 – Schéma de construction de la boîte-S de Zorro.

La constante 0xb2 est rajoutée ensuite pour enlever un point fixe, ce qui conduit à la boîte-S décrite par la table 4.2.

Le résumé des caractéristiques de la boîte-S ainsi obtenue ainsi que de celles des boîtes-S de l'AES et de PICARO est donné dans la table 4.1 (cette comparaison est celle faite par les concepteurs de Zorro dans le papier de sa spécification [GGNS13]). Rappelons ici que l'objectif des concepteurs de Zorro était de proposer une alternative à l'AES (même taille de bloc, même sécurité) qui soit plus performante que celui-ci lors de l'utilisation d'une technique de masquage : comparer leur boîte-S à celle de l'AES est donc tout naturel. Le chiffrement PICARO, quant à lui, est le premier chiffrement (mais aussi l'unique au moment de la conception de Zorro, du moins à notre connaissance) explicitement conçu pour être facile à masquer et représente donc le concurrent direct de Zorro, d'où sa prise en compte ici. On peut voir que les boîtes-S de PICARO et de Zorro nécessitent le même nombre de multiplications sécurisées, toutes deux sur 4 bits au lieu de 8 pour l'AES. Le calcul de la boîte-S de Zorro demande l'application de 4 matrices de diffusion supplémentaires (notées M sur la figure 4.3) ainsi que 4 élévations au carré (nécessaires au calcul de X^3). La boîte-S de l'AES nécessite des élévations au carré supplémentaires (7 au total) mais une seule application de matrice de diffusion. Le calcul du nombre de bits aléatoires nécessaires est moins direct. Ils proviennent en partie d'opérations de *rafraîchissement* des masques servant à s'assurer que ceux-ci restent indépendants entre eux. PICARO et Zorro nécessitent environ moitié moins de tels bits aléatoires. En ce qui concerne les propriétés cryptographiques, on peut observer que la boîte-S de Zorro possède une uniformité différentielle moins bonne que celle de PICARO et de l'AES, mais que son degré algébrique est égal à celui de la boîte-S de l'AES, alors que la boîte-S de PICARO n'a qu'un degré égal à 4.

	aléa (en bits)	nbr mult. sécurisées	autres opérations	propriétés cryptographiques		
				degré	unif. diff.	linéarité
AES	$16d^2 + 32d$	4 dans $\mathbb{F}_2^8$	7sq. + 1 matrice	7	4	32
PICARO	$8d^2 + 8d$	4 dans $\mathbb{F}_2^4$	2 sq.	4	4	68
`Zorro`	$8d^2 + 24d$	4 dans $\mathbb{F}_2^4$	4 sq. + 4 matrices	7	10	64

TABLE 4.1 – Comparaison des boîtes-S de l'AES, PICARO et `Zorro` vis-à-vis du schéma de masquage proposé par Rivain et Prouff [RP10]. Source : *Block Ciphers That Are Easier to Mask : How Far Can We Go ?* [GGNS13].

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	08	0c	03	06	06	04	05	06	05	04	0c	0c	04	03	05	03
00	b2	e5	5e	fd	5f	c5	50	bc	dc	4a	fa	88	28	d8	e0	d1
10	b5	d0	3c	b0	99	c1	e8	e2	13	59	a7	fb	71	34	31	f1
20	9f	3a	ce	6e	a8	a4	b4	7e	1f	b7	51	1d	38	9d	46	69
30	53	0e	42	1b	0f	11	68	ca	aa	06	f0	bd	26	6f	00	d9
40	62	f3	15	60	f2	3d	7f	35	63	2d	67	93	1c	91	f9	9c
50	66	2a	81	20	95	f8	e3	4d	5a	6d	24	7b	b9	ef	df	da
60	58	a9	92	76	2e	b3	39	0c	29	cd	43	fe	ab	f5	94	23
70	16	80	c0	12	4c	e9	48	19	08	ae	41	70	84	14	a2	d5
80	b8	33	65	ba	ed	17	cf	96	1e	3b	0b	c2	c8	b6	bb	8b
90	a1	54	75	c4	10	5d	d6	25	97	e6	fc	49	f7	52	18	86
a0	8d	cb	e1	bf	d7	8e	37	be	82	cc	64	90	7c	32	8f	4b
b0	ac	1a	ea	d3	f4	6b	2c	ff	55	0a	45	09	89	01	30	2b
c0	d2	77	87	72	eb	36	de	9e	8c	db	6c	9b	05	02	4e	af
d0	04	ad	74	c3	ee	a6	f6	c7	7d	40	d4	0d	3e	5b	ec	78
e0	a0	b1	44	73	47	5c	98	21	22	61	3f	c6	7a	56	dd	e7
f0	85	c9	8a	57	27	07	9a	03	a3	83	e4	6a	a5	2f	79	4f

TABLE 4.2 – Spécification de la boîte-S 8×8 de `Zorro`.

Fonction de tour

Le second angle d'attaque adopté par les concepteurs de `Zorro` pour minimiser le surcoût provenant du masquage fut de modifier la structure globale du chiffrement. Comme annoncé précédemment la fonction de tour de `Zorro` est basée sur celle de l'AES.

Elle se décompose en les fonctions suivantes (voir figure 4.4) :

1. *SubBytes** (SB^*) [6] : Application de la boîte-S de `Zorro` sur les octets 1 à 4 de l'état interne (correspondant à la première ligne)

6. Le symbole * a été introduit par les concepteurs pour différentier cette étape de celle de l'AES, notée *SubBytes* et abréviée en *SB*.

2. *AddConstant* (AC) : Ajout de la constante i (correspondant à l'indice de tour) aux octets 1 à 3, et ajout de $i \ll 3$ à l'octet 4
3. *ShiftRows* (SR) : Rotation circulaire de chaque ligne de l'état interne : chaque octet est décalé vers la gauche d'un nombre d'octets égal à l'indice de sa ligne
4. *MixColumns* (MC) : Application d'une matrice de diffusion sur chacune des colonnes de l'état interne

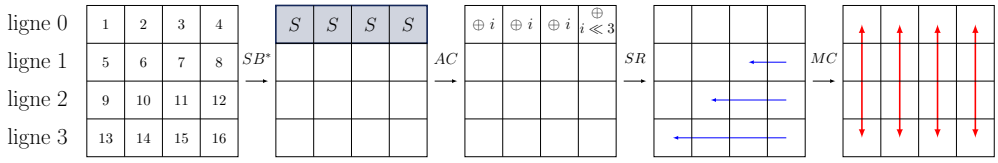


FIGURE 4.4 – Fonction de tour de Zorro : comme dans le cas de l'AES on représente l'état interne de 128 bits par une matrice de 4 octets sur 4 octets. Les opérations réalisées à chaque tour sont : l'application de boîtes-S sur la première ligne de l'état (SB^*), l'ajout de constantes (AC) puis les 2 opérations linéaires de l'AES (SR et MC).

Structure globale

La structure globale de Zorro, représentée à la figure 4.5, est similaire à celle de LED [GPPR11] : le chiffrement alterne 6 *étapes* (chacune formée par 4 fonctions de tour) avec des additions de la clef-maître K. Au total, la fonction de tour est appliquée 24 fois.

Ce nombre a été choisi suite à une analyse de sécurité approfondie considérant les attaques :

— Linéaires et différentielles
— Différentielles tronquées
— Meet-in-the-middle et bicliques
— Différentielles impossibles
— Algébriques
— Par rebond.

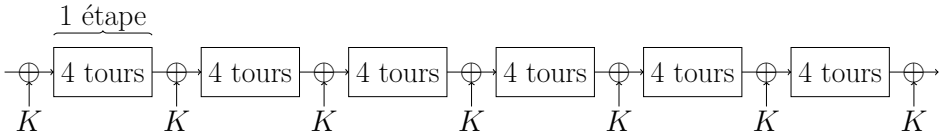


FIGURE 4.5 – Structure globale de Zorro : la fonction de tour est répétée 24 fois, et la clef-maître est additionnée à l'état interne tous les 4 tours. On appelle *étape* un ensemble de 4 tours.

Au total, et puisque le cadencement de clef de Zorro est réduit à l'identité, un chiffrement nécessite $4 \times 24 = 96$ boîtes-S. Ce nombre est à comparer à ce qui est nécessaire pour un chiffrement avec l'AES, qui s'élève à [7] : $16 \times 10 + 4 \times 10 = 200$ boîtes-S (16 dans chacun des 10 tours plus 40 dans l'algorithme de cadencement de clef). On a donc plus de 2 fois

7. on considère ici la version de l'AES avec une clef de la même taille, soit 128 bits.

moins d'applications de boîtes-S dans `Zorro` que dans l'AES et, qui plus est, celles-ci sont beaucoup plus simples à masquer, ce qui permet d'obtenir des performances bien meilleures (*cf.* la spécification de `Zorro` [GGNS13]).

4.2.2 Analyse de sécurité donnée par les auteurs : la technique des degrés de liberté

Lors de l'analyse de leur proposition, les auteurs ont fait face à plusieurs difficultés dues au fait que les techniques existantes (utilisant par exemple la propriété MDS de la fonction de diffusion) ne s'appliquent pas aux réseaux de substitution-permutation où seulement une partie de l'état interne entre dans les fonctions non-linéaires. Il est facile de voir qu'avec cette construction le nombre de boîtes-S actives augmente très lentement, et qu'il est même possible d'avoir une différence passant un tour complet avec probabilité 1 (il suffit de positionner celle-ci dans les 3 dernières lignes de l'état interne), ce qui n'est pas possible avec un SPN classique.

Pour combler ce manque, les auteurs ont fait appel à des techniques empruntées à l'analyse des fonctions de hachage et reposant sur la notion de *degré de liberté*. L'idée est de considérer une différence en entrée sur laquelle aucune condition ne pèse, puis de compter le nombre maximum de boîtes-S pouvant être laissées inactives. Pour cela, on considère que chaque condition que l'on veut satisfaire sur la transition effectuée par MC diminue d'autant le degré de liberté, jusqu'à ce que celui-ci vaille 0 et qu'aucune condition supplémentaire ne puisse être satisfaite. Avec cette technique, on obtient que le nombre maximum de tours pouvant être atteints par une caractéristique différentielle (sans dépasser le coût de la recherche exhaustive) s'élève à 14, et à 16 dans le cas d'un chemin linéaire. Le raisonnement suivi dans [GGNS13] est le suivant :

Le raisonnement par degrés de liberté consiste à examiner une paire de messages en entrée du chiffrement sur laquelle aucune condition ne pèse, donc de degré de liberté (en octets) de $16 + 16 = 32$. On considère ensuite que pour obtenir x boîtes-S inactives dans le tour suivant il faudra imposer autant de conditions (en nombre d'octets) sur le message donc que cela revient à réduire le degré de liberté initial d'autant.

Ce raisonnement modélise la situation que l'on aurait si on souhaitait résoudre le problème avec un système d'équations : on considère au départ une paire de messages sur laquelle aucune condition ne pèse, donc avec 32 variables (octets) *libres*. Rendre inactive une boîte-S correspond à contraindre la paire de 4 octets en entrée de l'opération *MixColumns* précédente de sorte que l'octet de sortie en question soit nul. De la sorte, chaque boîte-S inactive impose une équation sur les octets, donc fixe une relation entre les variables en entrée de la différence.

Dans ce modèle, chaque inactivation supplémentaire est supposée consommer un degré de liberté [a], et on admet qu'il existe une solution non triviale s'il reste au minimum un degré de liberté. Le raisonnement suivi par les concepteurs de `Zorro` dans [GGNS13] est comme suit.

Étant donné que la clef maître fait 128 bits, une caractéristique est exploitable dans une attaque si sa probabilité est strictement supérieure à 2^{-128}. Comme l'uniformité différentielle de la boîte-S de `Zorro` vaut 10 cela signifie que celle-ci peut avoir un maximum de nb_a boîtes-S actives, où nb_a vérifie : $(\frac{10}{256})^{nb_a} > 2^{-128}$, soit $nb_a < 27.4$ et donc que la caractéristique recherchée doit activer strictement moins de 28 boîtes-S.

On recherche donc le nombre de tours r maximum qu'une caractéristique de probabilité strictement supérieure à 2^{-128} peut couvrir sachant qu'on possède 32 degrés de liberté pouvant permettre de rendre inactives des boîtes-S. Si on note par x_i le nombre de degrés de liberté *utilisés* au tour i ($1 \leq i \leq r$), cela signifie qu'on cherche r vérifiant :

$$\sum_{i=1}^{r} x_i < 32 \quad \text{et} \quad 4 \times r - \sum_{i=1}^{r} x_i < 28$$

Si on note $\bar{x}$ le nombre moyen de degrés de liberté utilisés par tour, on peut réécrire ce système en :

$$r \times \bar{x} < 32 \quad \text{et} \quad 4 \times r - r \times \bar{x} < 28$$

Ce qui implique un nombre de tours (r) maximum égal à 14 avec un nombre de boîtes-S actives comprises entre 27 et 25.

a. On verra que c'est cette hypothèse qui s'avère incorrecte et qui a des conséquences désastreuses pour Zorro.

De façon surprenante ces arguments se sont avérés insuffisants puisque deux cryptanalyses de la version complète (utilisant des techniques différentielles et linéaires) furent annoncées peu de temps après la parution de Zorro, ainsi qu'une attaque en clefs faibles valide pour 2^{64} des 2^{128} clefs. Les complexités de ces résultats ainsi que des nôtres sont données dans la table 4.3.

Source	C_T	C_D	C_M	Technique
[GNPW13]	$2^{54.25}$	$2^{54.25}$ CP	$2^{54.25}$	Internal Differential
[WWGY14]	2^{112}	2^{112} CP	negligible	Différentielle
[RASA14]	$\approx 2^{55}$	$2^{55.12}$ CP	2^{17}	Différentielle
[RASA14]	$2^{57.85}$	$2^{45.44}$ KP	2^{17}	Linéaire
Sec. 4.3.4	2^{45}	$2^{41.5}$ CP	2^{10}	Différentielle

KP - Clair connu, CP - Clair choisi

TABLE 4.3 – Précédents résultats de cryptanalyse sur la version complète de Zorro et résultats obtenus avec notre technique.

4.2.3 Explication de l'échec de la technique des degrés de liberté dans le cas de Zorro

Pour comprendre pourquoi l'argument des degrés de liberté n'a pas fonctionné, étudions les attaques différentielles proposées par Yanfeng Wang et al. [WWGY14] et Rasoolzadeh et al. [RASA14].

Leur point de départ est la recherche d'une caractéristique itérative sur 4 tours (une étape) mettant à profit la propriété de la matrice M utilisée dans l'opération *MixColumns* :

$$M^4 = Id$$

combiné au fait que *ShiftRows* soit aussi d'ordre 4.

Dans l'attaque présentée par Yanfeng Wang et al. la différence d'entrée est de la forme :

$$\Delta_{in} = \{0,0,0,0,\ A,A,A,A,\ 0,0,0,0,\ B,B,B,B\}$$

où A et B représentent des éléments de $\mathbb{F}_2^8$ (on donne ici les octets dans l'ordre défini sur la figure 4.4). Pour une telle différence et de façon plus générale pour une différence dont les lignes sont toutes identiques l'opération *ShiftRows* n'aura aucun effet sur la différence et l'opération *MixColumns* conservera l'égalité des colonnes. Si on arrive à s'assurer que les boîtes-S se comportent comme l'identité sur les différences (c'est-à-dire si leurs différences d'entrée sont égales à leurs différences de sortie), alors une différence en entrée d'une étape de `Zorro` sera égale à sa différence en sortie, et la caractéristique sera itérative :

$$\begin{aligned}
\Delta_{out} &= MC(SR(AC(SB^*(MC(SR(AC(SB^*(MC(SR(AC(SB^*(MC(SR(AC(SB^*(M))))))))))))))) \oplus \\
&\quad MC(SR(AC(SB^*(MC(SR(AC(SB^*(MC(SR(AC(SB^*(MC(SR(AC(SB^*(M+\Delta_{in})))))))))))))))) \\
&= MC(SR(MC(SR(MC(SR(MC(\underbrace{SR(\Delta_{in})}_{=\Delta_{in}})))))))) \\
&= MC(SR(MC(SR(MC(\underbrace{SR(MC(\Delta_{in}))}_{=MC(\Delta_{in})})))))) \\
&= \dots \\
&= MC^4(\Delta_{in}) \\
&= \Delta_{in}
\end{aligned}$$

Cette observation est généralisée dans l'attaque présentée par Rasoolzadeh et al. [RASA14] pour laquelle la différence en entrée est choisie de la forme {*0,0,0,0, A,B,A,B, C,D,C,D, E,F,E,F*} (voir figure 4.6) où les variables A à F représentent des éléments de $\mathbb{F}_2^8$:

- la première ligne de la différence est choisie égale à zéro pour qu'aucune boîte-S ne soit active au premier tour,
- les colonnes 1 et 3 ainsi que 2 et 4 sont égales pour annuler l'effet de *ShiftRows* sur 4 tours de `Zorro` (et conserver l'égalité des colonnes à chaque tour),
- les valeurs possibles des 6 octets de différence sont choisies de sorte que les tours 2 et 3 soient inactifs (comme représenté à la figure 4.6).

Les auteurs écrivent ensuite les équations que doivent satisfaire les 6 octets de la différence d'entrée de sorte que les boîtes-S du tour 2 et 3 soient inactives (à commencer par $E = 3A \oplus D$ pour que la première boîte-S après le premier *MixColumns* soit inactive) puis résolvent le système. Cette résolution leur permet d'exprimer toutes les variables intervenant dans les 4 tours représentés en figure 4.6 en fonction de A et B [RASA14] :

$$\begin{aligned}
C &= A \oplus B, & D &= A \oplus B, & E &= 2A \oplus B, \\
F &= A \oplus 2B, & G &= 2A \oplus 3B, & H &= 3A \oplus 2B, \\
I &= A \oplus 5B, & J &= 5A \oplus B, & K &= 3A \oplus 4B, \\
L &= 4A \oplus 3B, & M &= A \oplus 8B, & N &= 8A \oplus B, \\
O &= 13(A \oplus B), & P &= 13(A \oplus B), & Q &= 10A \oplus B, \\
R &= A \oplus 10B, & S &= 20A \oplus 4B, & T &= 4A \oplus 20B, \\
U &= 6A \oplus 31B, & V &= 31A \oplus 6B, & W &= 17A \oplus 5B, \\
X &= 5A \oplus 17B, & Y &= 7A \oplus 24B, & Z &= 24A \oplus 7B,
\end{aligned}$$

ce qui indique que l'ensemble de solutions est de taille 2^{16}. Il est possible d'imposer une condition supplémentaire et de fixer l'octet T (ou S) à zéro. L'ajout de cette équation détermine A en fonction de B et permet de décrire l'ensemble de toutes les caractéristiques sur 4 tours itératives avec 2 boîtes-S actives possédant cette structure.

Pour que la caractéristique soit itérative il faut s'assurer (comme pour l'attaque précédente) que l'opération SB^* agisse comme l'identité et donne $S \rightarrow_S S$ et $T \rightarrow_S T$. En se référant à la table des différences de la boîte-S de `Zorro` on peut voir que les meilleurs choix pour S et/ou T sont `0x7b`, `0xea` et `0xf7` qui conduisent tous les 3 à des transitions de probabilité 6/256.

Jusqu'à la fin du tour 4, le raisonnement par degrés de liberté et la résolution des équations donnent les mêmes résultats, mais la première analyse devient fausse si on la poursuit aux tours 5 et suivants. Le problème provient du fait que même si annuler une boîte-S au tour 5 impose 8 équations binaires supplémentaires à satisfaire, celles-ci ne *dépensent pas* de degrés de liberté car elles ne sont pas indépendantes des équations précédentes. En effet, étant donné que la caractéristique est itérative, la différence en entrée du tour 5 est égale à celle en entrée du tour 1 (soit $\{0,0,0,0,\quad A,B,A,B,\quad C,D,C,D,\quad E,F,E,F\}$) donc les 4 boîtes-S de ce tour-ci sont inactives avec probabilité 1. Comme de plus A, B, C, D, E et F on été choisis de sorte que l'opération *MixColumns* résulte en un état inactif sur la première ligne, on aura nécessairement inactivité des 4 boîtes-S du tour 6, et ainsi de suite jusqu'au tour 8 où la différence d'entrée sera de $\{S,T,S,T,\quad U,V,U,V,\quad W,X,W,X,\quad Y,Z,Y,Z\}$. Sous condition que les boîtes-S assurent les mêmes transitions $S \rightarrow_S S$ et $T \rightarrow_S T$ que précédemment, ce phénomène se reproduira dans les 4 tours suivants.

Ainsi, choisir convenablement les différences A à F permet de s'assurer que les 3 premiers tours de la caractéristique représentée à la figure 4.6 seront suivis avec probabilité 1. Le seul événement probabiliste de l'étape se situe dans le quatrième tour où les 4 boîtes-S doivent laisser la différence inchangée. Sous la seule condition que ces 4 transitions se passent favorablement, on aura une caractéristique itérative avec peu de boîtes-S actives.

C'est précisément ce type de situation qui est oublié par l'analyse donnée par les auteurs de `Zorro` : avec une caractéristique itérative comme celle proposée par Yanfeng Wang et al. [WWGY14] et Rasoolzadeh et al. [RASA14], obtenir autant de boîte-S inactives à l'étape 2 qu'à l'étape 1 est immédiat et ne diminue pas le nombre de degrés de liberté. De façon plus générale si plusieurs conditions imposées à la différence d'entrée ne sont pas indépendantes l'analyse reposant sur le calcul des degrés de liberté échoue.

Les résultats obtenus par Yanfeng Wang et al. [WWGY14] puis par Rasoolzadeh et al. [RASA14] montrent que `Zorro` est extrêmement vulnérable aux attaques différentielles et linéaires. Cependant, ils ne répondent pas à la question plus générale de savoir si toutes les constructions de type PSPN possèdent un problème structurel face à ces attaques ou s'il est possible de construire une instance plus résistante [8].

4.3 Description de nos outils d'analyse et d'attaque des PSPN

C'est précisément cette question de généralisation qui motiva nos travaux [BDD+15] et qui nous amena à développer des techniques d'analyse des constructions PSPN. Notre première contribution est un outil de recherche de caractéristiques différentielles et linéaires de forte

8. En effet, si on change légèrement la définition des étapes linéaires ou l'emplacement de boîtes-S de `Zorro`, les attaques de Yanfeng Wang et al. [WWGY14] et de Rasoolzadeh et al. [RASA14] ne s'appliquent plus.

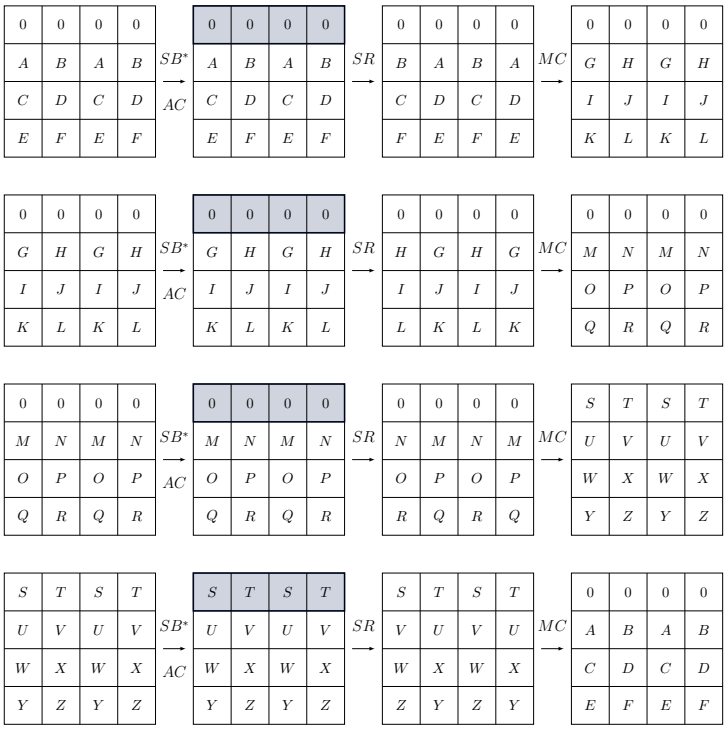


FIGURE 4.6 – Forme générale de la caractéristique utilisée par [RASA14].

probabilité. Lorsqu'on applique celui-ci à une construction particulière, il permet de calculer le nombre minimum de boîtes-S actives d'une caractéristique sur un nombre de tours donné et ainsi d'apporter une première évaluation de la résistance du chiffrement aux attaques différentielles et linéaires de base. Contrairement à la technique heuristique proposée par les concepteurs de `Zorro` ou à certaines techniques s'appliquant aux chiffrements SPN conventionnels (comme celles basées sur l'optimisation linéaire en nombres entiers, en anglais MILP pour *Mixed Integer Linear Programming* [MWGP11]), l'évaluation retournée par notre algorithme est un minimum qui est atteint. De plus, cette recherche possède un temps d'exécution raisonnable jusqu'à un assez grand nombre de tours.

Dans le cas où ce premier algorithme retourne des caractéristiques différentielles ou linéaires de probabilité suffisamment élevée (comme c'est le cas pour `Zorro`), on pourra utiliser ces biais pour monter une cryptanalyse différentielle ou linéaire. Nous proposons un second algorithme permettant de faire ceci de façon efficace en exploitant là encore la structure spécifique des PSPN, plus précisément en optimisant l'étape de recouvrement de clef. Au lieu de se contenter de réaliser une attaque sur le dernier tour (et donc de laisser la différence se propager librement sur un seul tour), notre technique exploite la grande linéarité de la construction pour analyser plus de tours sans augmenter la complexité de l'attaque.

4.3.1 Algorithme de recherche de caractéristiques différentielles de forte probabilité

L'algorithme de recherche de caractéristiques que nous présentons dans [BDD+15] se fonde sur l'utilisation d'une représentation dite par *motifs* (ou *patterns*) permettant de regrouper

et d'analyser ensemble un groupe de caractéristiques.

Définition 4.2. *(Motifs) Les motifs sont une représentation simplifiée d'une caractéristique ne retenant comme information que le motif d'activité de celle-ci : pour chaque mot (typiquement pour chaque octet) on ne gardera qu'une information binaire indiquant si sa différence est nulle ou non.*

Pour notre algorithme nous nous intéressons aux motifs d'activité de l'état en entrée de l'opération SB^* et même plus précisément uniquement à la présence ou à l'absence de différence dans les octets qui s'apprêtent à être modifiés par le passage par une boîte-S. Ainsi, notre motif ne correspondra pas au motif d'activité de l'état interne complet comme pour les algorithmes précédents, mais uniquement à celui de certains octets de l'état (voir la figure 4.7 dans le cas de `Zorro`).

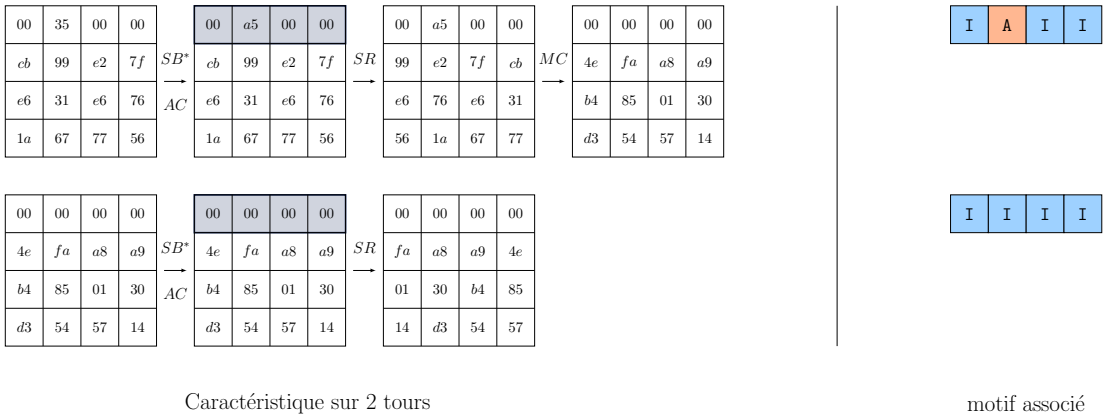


FIGURE 4.7 – Exemple de caractéristique sur 2 tours avec son motif associé. On s'intéresse uniquement à l'information d'activité (A) ou d'inactivité (I) des octets entrant dans les boîtes-S.

L'algorithme que nous avons développé dans [BDD+15] s'applique à la fois à la recherche de caractéristiques différentielles et à la recherche de caractéristiques linéaires. Dans les deux cas son déroulement est très similaire, donc pour faciliter nos explications nous prenons la liberté de ne décrire ici que le cas différentiel.

Notations

On considérera dans la suite un chiffrement PSPN de type AES agissant sur des états de 128 bits représentés par des matrices de 4 octets sur 4 octets. Pour que notre description soit la plus générale possible, on introduit la variable t pour représenter le nombre de boîtes-S 8×8 appliquées à chaque étape non linéaire. La structure de chaque tour des chiffrements considérés est alors la suivante (*cf.* la figure 4.8) :

— **Étape non-linéaire :** La boîte-S 8×8 transforme t octets sur les 16 que comporte l'état [9]. Cette boîte-S se doit bien entendu d'être une bijection.

9. Avec cette notation, $t = 16$ pour l'AES et $t = 4$ pour `Zorro`.

— **Étape linéaire :** Aucune restriction n'est imposée sur la structure de l'étape linéaire utilisée dans la fonction de tour [10], si ce n'est qu'elle doit là aussi être bijective. On note L cette étape.
— **Étape d'addition de clef :** Dans cette étape la clef de tour est additionnée à l'état interne par `XOR`. On considère que la clef-maître fait 128 bits mais on n'émet aucune restriction sur la fréquence d'addition des clefs de tours (cela peut être à chaque tour ou tous les 4 tours comme pour `Zorro` par exemple).

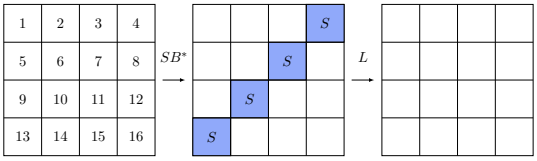


FIGURE 4.8 – Exemple de PSPN avec $t = 4$ boîtes-S par tour.

Utilisation des motifs et justification de la faisabilité de leur étude

L'outil que nous avons développé a pour objectif de déterminer si une construction PSPN donnée est sensible aux attaques différentielles et linéaires de base. On souhaite donc qu'il puisse identifier si le chiffrement possède des caractéristiques de forte probabilité, ce qui est équivalent à pouvoir déterminer s'il possède des caractéristiques avec peu de boîtes-S actives.

Supposons par exemple que l'on souhaite étudier un PSPN avec t boîtes-S par tour et que l'on souhaite analyser r tours de cette construction. La première étape consiste à fixer une quantité a correspondant au nombre maximal de boîtes-S actives que l'on s'autorise (on peut par exemple choisir cette quantité en établissant qu'au-delà de a boîtes-S actives la probabilité de la caractéristique sera trop faible pour représenter une menace pour le chiffrement).

Pour déterminer s'il existe des caractéristiques qui suivent cette contrainte, l'algorithme va considérer les uns après les autres tous les motifs avec a boîtes-S actives et réaliser un traitement pour décider s'ils sont possibles.

La faisabilité de cette étude repose sur les deux remarques suivantes :

1. **Le nombre de motifs différents à étudier est petit.** Si le nombre de boîtes-S présentes dans chaque tour est petit et qu'on limite notre recherche à des caractéristiques possédant peu de boîtes-S actives alors le nombre de configurations à étudier reste raisonnable pour un assez grand nombre de tours. En effet il faudra étudier au plus [11]

$$\binom{t \cdot r}{\leq a} \triangleq \sum_{i=0}^{a} \binom{t \cdot r}{i}$$

motifs pour traiter toutes les caractéristiques possédant au plus a boîtes-S actives parmi les tr boîtes-S que compte le chiffrement. Par exemple pour chercher les caractéristiques sur $r = 9$ tours avec au plus $a = 4$ boîtes-S actives sur un chiffrement

10. On pourra même considérer que celle-ci varie à chaque tour, ou — comme c'est le cas pour l'AES — qu'elle est différente dans le dernier tour.
11. Nous verrons en section 4.3.2 comment éviter d'avoir à parcourir tous les motifs.

possédant $t = 4$ boîtes-S par tour (comme `Zorro`) on devra étudier au plus $\binom{36}{\leq 4} \approx 2^{17}$ motifs différents.

2. **L'ensemble des caractéristiques suivant un motif donné peut être étudié efficacement.** Pour un motif donné, l'algorithme retournera un ensemble d'équations linéaires [12] auquel les caractéristiques doivent se conformer. Pour déterminer ces dernières, il faudra alors itérer sur ce sous-espace linéaire. Cette opération se résume à des opérations d'algèbre linéaire donc peut être réalisée efficacement.

Première intuition

Supposons que l'on souhaite étudier un PSPN avec t boîtes-S par tour et que l'on veuille tester l'existence de caractéristiques différentielles avec moins de a boîtes-S actives sur r tours (les paramètres t, a et r sont fixés).

L'algorithme étudie les uns après les autres les $\binom{t \cdot r}{\leq a}$ motifs possibles. Lors de l'initialisation, 128 variables binaires sont introduites pour représenter les 128 bits de la différence entre les 2 messages clairs en entrée du chiffrement. L'idée est ensuite de suivre l'évolution de cette différence lors du calcul des r tours, d'observer ce que le motif implique comme contraintes sur ces variables et finalement d'en déduire s'il existe des caractéristiques suivant le motif de départ. Plus en détail, pour établir l'ensemble des contraintes imposées par le motif on partira de la différence des messages clairs exprimée par 128 variables et on la déroulera tour après tour en suivant les règles suivantes :

— Pour calculer l'évolution de la différence au travers des étapes linéaires L on applique simplement L à la différence symbolique.

— Pour calculer l'évolution de la différence au travers des étapes non-linéaires (i.e. des boîtes-S) on se réfère au motif : si celui-ci indique qu'une boîte-S est inactive cela nous indique que l'octet de la différence entrant dans cette boîte-S est nul et on obtient 8 équations linéaires à vérifier [13]. Si au contraire le motif indique que la boîte-S est active on introduira 8 nouvelles variables binaires pour représenter la différence en sortie. Ainsi, l'expression symbolique de la différence sera toujours une fonction linéaire de la différence en entrée et de ces variables intermédiaires. On prendra aussi note de la différence symbolique en entrée et en sortie de la boîte-S.

Chaque boîte-S active du motif impose d'introduire 8 nouvelles variables binaires, donc après avoir analysé les t tours du PSPN on aura un ensemble d'équations exprimées avec au plus $128 + 8a$ variables. Comme les autres boîtes-S sont forcément inactives elles fourniront au minimum $8(t \cdot r - a)$ équations linéaires (indépendantes ou non). Si a est suffisamment petit comparé à $t \cdot r$ (c'est-à-dire quand la majorité des boîtes-S sont inactives) la dimension de l'espace linéaire dans lequel les caractéristiques solutions résident est petite. On peut alors itérer sur celui-ci pour obtenir les valeurs des variables binaires : à la fois des 128 variables initiales représentant la différence entre les messages clairs mais aussi de celles introduites pour linéariser les sorties des boîtes-S actives. Pour s'assurer que la caractéristique obtenue est véritablement solution, on s'assure que les transitions qu'elle définit pour les boîtes-S actives sont des transitions valides en se référant à la DDT de celle-ci. Cette dernière opération

12. Nous allons expliquer ceci en détail plus loin.
13. Il s'agit des équations exprimant le fait que ces 8 bits de la différence sont nuls.

nous permet d'éliminer des caractéristiques impossibles.

Description détaillée

Nous donnons ici la description formelle de l'algorithme de recherche de caractéristiques. Comme précédemment, on considère que l'on a fixé les paramètres a (nombre de boîtes-S actives maximal) et r (nombre de tours analysés) pour lesquels on étudie un PSPN avec t boîtes-S par tour.

L'algorithme se décompose en deux grandes étapes, réalisées pour chacun des motifs possibles avec moins de a boîtes-S par tour. La première consiste à calculer le sous-espace linéaire dans lequel résident les caractéristiques solutions du motif et la seconde détermine leurs valeurs explicites.

Calcul du sous-espace linéaire solution d'un motif. Pour calculer le sous-espace linéaire défini par le motif, l'idée est de suivre l'évolution de la différence en entrée (exprimée par 128 variables binaires) lors du calcul des r tours de chiffrement.

De façon concrète, l'algorithme va manipuler la représentation symbolique de la différence, c'est-à-dire un ensemble de 128 combinaisons linéaires représentant chacune un bit de l'état interne. On nommera ST_i l'expression obtenue à la fin du tour i. La différence ST_0 (correspondant à la différence sur les clairs) sera initialisée par 128 variables (1 pour chaque bit).

De la même manière, l'algorithme manipulera et maintiendra à jour un système d'équations correspondant aux contraintes linéaires que les caractéristiques doivent satisfaire. Au début de l'analyse, celui-ci sera initialisé à l'ensemble nul et sera noté $\mathcal{E}_0$. On mettra à jour ce système grâce à l'information déduite du motif, et on notera $\mathcal{E}_i$ le système obtenu après i tours. L'ensemble des conditions que devront respecter les caractéristiques pour suivre le motif sont celles exprimées par $\mathcal{E}_r$.

L'algorithme suivant décrit comment prendre en compte l'information donnée par le motif au tour $i+1$ lorsqu'on a analysé le chiffrement jusqu'au tour i, c'est-à-dire comment calculer ST_{i+1} et $\mathcal{E}_{i+1}$ à partir de ST_i et $\mathcal{E}_i$.

Extension sur 1 tour de la linéarisation :

1. Allouer et initialiser ST_{i+1} et $\mathcal{E}_{i+1}$ par : $ST_{i+1} \leftarrow ST_i, \mathcal{E}_{i+1} \leftarrow \mathcal{E}_i$.
2. Pour chaque boîte-S S du tour i :
   (a) Si le motif indique que S est inactive, ajouter au système $\mathcal{E}_{i+1}$ les 8 équations correspondant à annuler les 8 bits en entrée de S et mettre à zéro les 8 bits correspondants de ST_{i+1}. Si l'ensemble de solutions de $\mathcal{E}_{i+1}$ devient trivial (c'est-à-dire si $\mathcal{E}_{i+1}$ n'admet pas de solution autre que celle entièrement nulle, donc qu'aucune caractéristique n'est solution), retourner ST_{i+1} et $\mathcal{E}_{i+1}$ égaux à `NULL` et terminer.
   (b) Si le motif indique que S est active, remplacer les 8 bits correspondants de ST_{i+1} par 8 nouvelles variables et maintenir $\mathcal{E}_{i+1}$ inchangé.
3. Calculer le passage de l'étape linéaire sur la différence formelle : $ST_{i+1} \leftarrow L(ST_{i+1})$.

Avec cet algorithme et pour un motif donné il devient facile de calculer le sous-espace dans lequel résident les caractéristiques solutions :

Calcul du sous-espace défini par un motif :

1. Initialiser la différence des clairs ST_0 avec 128 nouvelles variables binaires et le système d'équations $\mathcal{E}_0$ à un ensemble vide.
2. Exécuter le précédent *algorithme d'extension sur 1 tour de la linéarisation* pour $i = 0$ à $i = r - 1$. Si celui-ci renvoie que ST_{i+1} et $\mathcal{E}_{i+1}$ sont `NULL`, retourner `NULL` et terminer.
3. Calculer le sous-espace des solutions de $\mathcal{E}_r$ et en déduire une base B décrivant les caractéristiques solutions (on note b le nombre de variables (binaires) libres de cette base). Renvoyer les expressions des 128 variables correspondant aux bits de ST_0, ainsi que les expressions des (au plus) $16 \cdot a$ bits représentant les entrées et sorties des (au plus) a boîtes-S actives.

Élimination des caractéristiques invalides. Pour obtenir les caractéristiques effectives, on parcourt l'ensemble des valeurs possibles pour les b variables libres et on en déduit les valeurs des bits obtenus en sortie de l'algorithme précédent. On s'assure ensuite que les caractéristiques obtenues sont possibles en vérifiant dans la DDT de la boîte-S du chiffrement que les transitions réalisées par les boîtes-S actives sont possibles (i.e. que leur probabilité est strictement supérieure à 0).

Élimination des caractéristiques invalides :

1. Pour chacune des 2^b valeurs possibles pour les variables libres :
   (a) Pour chaque transition de boîte-S :
      i. Calculer les différences d'entrée et de sortie de la boîte-S.
      ii. Vérifier dans la DDT que la transition est possible, et dans la négative retourner à l'étape 1.
   (b) Retourner la caractéristique complète.

Il est possible d'améliorer ce dernier algorithme en faisant en sorte (lors de la simplification du système linéaire $\mathcal{E}_r$) que les variables libres correspondent à des entrées et sorties de boîtes-S actives, et ainsi de ne considérer comme valeurs des bits libres que des valeurs correspondant à des transitions possibles (i.e. au lieu de fixer les b bits puis de vérifier la DDT, on se sert de la DDT pour choisir les b bits, ce qui élimine beaucoup de tests). Cette optimisation est particulièrement utile lorsque le nombre de variables libres est important.

Analyse de complexité. Soit T_{node} la complexité moyenne du calcul du sous-espace linéaire correspondant à un motif particulier (ne comprenant donc pas l'étape d'itération sur les variables libres ni l'élimination des caractéristiques invalides). On peut estimer la complexité de la recherche (complète) de caractéristiques par la formule :

$$\binom{t \cdot r}{\leq a} \cdot T_{node} + \mathsf{SOL}$$

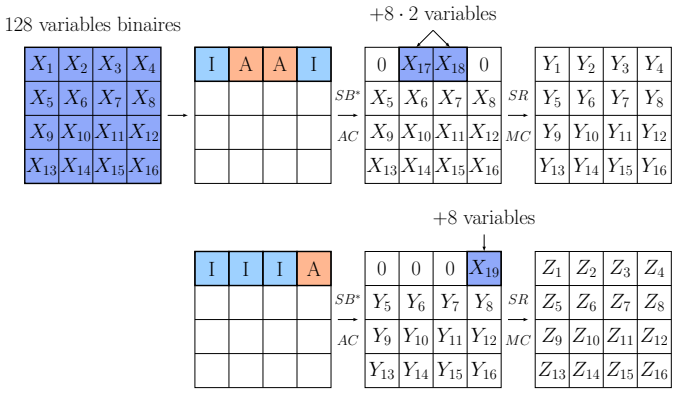


FIGURE 4.9 – Représentation schématique d'un exemple de déroulement de recherche de caractéristiques. On étudie ici $r = 2$ tours du chiffrement `Zorro` et plus particulièrement le motif $IAAI - IIIA$. On commence par exprimer la différence d'entrée avec 128 variables binaires et on suit la propagation de la différence tout au long des tours : si le *motif* indique une boîte-S active on introduit 8 nouvelles variables binaires, sinon on prend note de l'équation imposée et on fixe à zéro son octet de sortie.

où SOL correspond au nombre total de solutions obtenues en itérant sur les variables libres, avant élimination des caractéristiques invalides [14].

Cette formule peut être réécrite en fonction du nombre de caractéristiques effectivement retournées par l'algorithme : si on note Out ce nombre, comme la probabilité qu'une transition valide vaut (au moins) $2^{-1.5}$, on aura la relation : $\mathsf{Out} \geq \mathsf{SOL} \cdot 2^{-1.5 \cdot a}$, donc la complexité de l'algorithme peut être majorée par :

$$\binom{t \cdot r}{\leq a} \cdot T_{node} + \mathsf{Out} \cdot 2^{1.5 \cdot a}.$$

Si le nombre de caractéristiques retournées (Out) est de taille raisonnable, la complexité de l'algorithme est proportionnel à $\binom{t \cdot r}{\leq a}$.

4.3.2 Optimisation de l'algorithme de recherche de caractéristiques : recherche par préfixe commun

L'optimisation que nous décrivons ici (qui est celle que nous avons implémentée) repose sur la remarque suivante : parmi les $\binom{t \cdot r}{\leq a}$ motifs étudiés par l'algorithme, plusieurs partagent un même préfixe (par exemple ont le même motif d'activité sur le premier tour) ce qui implique que les mêmes calculs sont réalisés plusieurs fois (dans notre exemple le calcul de ST_1 et de $\mathcal{E}_1$). Pour diminuer la complexité de l'algorithme et limiter les calculs effectués en double, l'idée est alors d'organiser la recherche par préfixe commun sous forme d'arbre de recherche que l'on parcourra avec un parcours en profondeur (ou *DFS* pour *Depth First Search*). En organisant notre recherche de cette manière on pourra écarter tout un ensemble de motifs si on identifie que leur préfixe commun n'a pas de solution, et ainsi accélérer considérablement

14. Étant donné que l'analyse d'une solution est très simple on considère que son coût est unitaire.

le temps d'exécution de l'algorithme [15].

La structure de l'arbre de recherche utilisé pour cette optimisation est représenté à la figure 4.10. On suppose encore une fois que les paramètres a, t et r sont fixés, et ils constituent les variables globales de notre algorithme. La profondeur maximale de l'arbre de recherche utilisé est $t \cdot r$ et chaque nœud peut avoir jusqu'à deux fils : le fils gauche correspond au motif ayant la boîte-S suivante active alors que le fils droit correspond au motif ayant la boîte-S suivante inactive. Comme le nombre maximal de boîtes-S actives a est fixé (et relativement petit) le fils gauche n'existera pas pour tout nœud.

Le parcours en profondeur est réalisé par une série d'appels récursifs prenant en paramètre :

— l'indice de tour i,
— l'indice de la boîte-S courante dans le tour i (correspondant à un entier $s \in [0, t-1]$),
— le nombre de boîtes-S actives jusqu'ici dans le préfixe, noté ca,
— la valeur symbolique de la différence entre les états internes ST_i et
— le système d'équations linéaires régissant les caractéristiques suivant le préfixe $\mathcal{E}_i$.

On note $\mathsf{PPS}(i, s, \mathsf{ca}, ST_i, \mathcal{E}_i)$ un tel appel.

Le tout premier appel à l'algorithme se fait avec les paramètres $i = 0$, $s = 0$, $\mathsf{ca} = 0$, ainsi que (comme pour l'algorithme précédent) ST_0 initialisé à 128 variables binaires et $\mathcal{E}_0$ qui correspond à un système d'équations vide.

Le déroulement de $\mathsf{PPS}(i, s, \mathsf{ca}, ST_i, \mathcal{E}_i)$ est le suivant :

$\mathsf{PPS}(i, s, \mathsf{ca}, ST_i, \mathcal{E}_i)$ **:**

1. Si $i = r$ (c'est-à-dire si les r tours ont été traités), le système $\mathcal{E}_r$ obtenu correspond aux équations que les caractéristiques doivent vérifier pour suivre le motif défini par le chemin de la racine à la feuille courante. On peut alors déterminer la base B des caractéristiques solutions puis la filtrer par le même algorithme que celui utilisé dans la version précédente, puis retourner les solutions.
2. Allouer un nœud (le fils droit) correspondant au cas où la boîte-S d'indice s est inactive. Ajouter 8 équations au système $\mathcal{E}_i$ pour rendre compte de l'annulation des 8 bits de ST_i en entrée de la boîte-S. On appelle l'ensemble solution de ce nœud OUT_1.
   — Si la dimension de l'ensemble de solutions de $\mathcal{E}_i$ est 0 (c'est-à-dire si aucune caractéristique non-nulle n'est solution), supprimer le nœud courant et poser $\mathsf{OUT}_1 = \emptyset$
   — Si la dimension de l'ensemble de solutions de $\mathcal{E}_i$ est non nulle et que $s = t-1$ (ce qui indique qu'on a traité toutes les boîtes-S du tour i), calculer la valeur de la différence à la fin du tour en appliquant l'étape linéaire à ST_i : $ST_{i+1} = L(ST_i)$. Définir $\mathcal{E}_{i+1} = \mathcal{E}_i$ puis faire appel à $\mathsf{PPS}(i+1, 0, \mathsf{ca}, ST_{i+1}, \mathcal{E}_{i+1})$. On définit OUT_1 comme le résultat de cet appel.
   — Si la dimension de l'ensemble de solutions de $\mathcal{E}_i$ est non nulle et que $s < t-1$, faire appel à $\mathsf{PPS}(i+1, 0, \mathsf{ca}, ST_{i+1}, \mathcal{E}_{i+1})$. On définit OUT_1 comme le résultat

15. Comme ces cas d'éliminations sont assez difficiles à prévoir, il est compliqué de fournir une estimation de la complexité de l'algorithme.

de cet appel.

3. Si $\mathsf{ca} = a$ (c'est-à-dire si on a atteint le nombre maximum de boîtes-S actives) retourner OUT_1.
4. Si $\mathsf{ca} < a$ allouer un nœud (le fils gauche) correspondant au cas où la boîte-S d'indice s est active. Remplacer dans ST_i les expressions des 8 bits correspondant à la boîte-S courante par 8 nouvelles variables. On appelle l'ensemble solution de ce nœud OUT_2.
   — Si $s = t - 1$ (c'est-à-dire qu'on a traité toutes les boîtes-S du tour i), calculer la valeur de la différence à la fin du tour en appliquant l'étape linéaire à ST_i : $ST_{i+1} = L(ST_i)$. Définir $\mathcal{E}_{i+1} = \mathcal{E}_i$ puis faire appel à $\mathsf{PPS}(i+1, 0, \mathsf{ca}+1, ST_{i+1}, \mathcal{E}_{i+1})$ définir OUT_2 comme le résultat de cet appel.
   — Si $s < t-1$ faire un appel à $\mathsf{PPS}(i, s+1, \mathsf{ca}+1, ST_i, \mathcal{E}_i)$ et définir OUT_2 comme le résultat de cet appel.
5. Retourner $\mathsf{OUT}_1 \bigcup \mathsf{OUT}_2$.

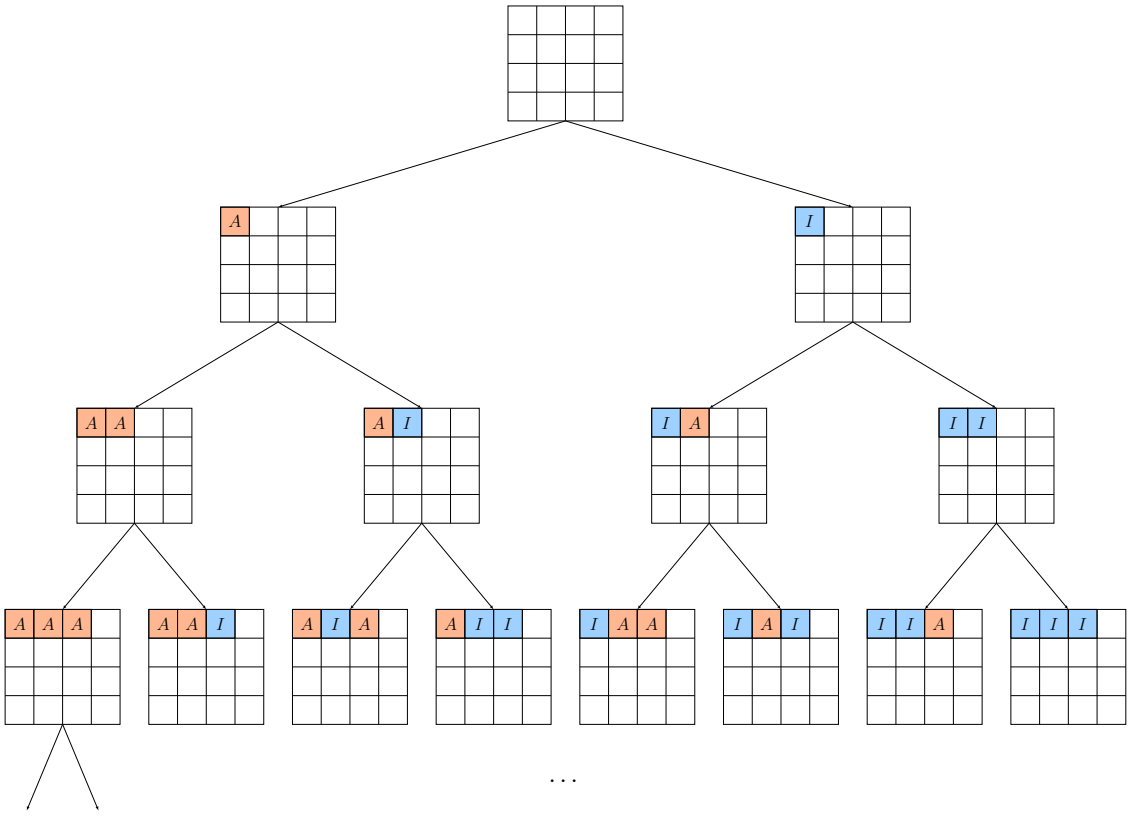


FIGURE 4.10 – Exemple d'arbre de recherche parcouru pour la recherche de caractéristiques dans le cas de `Zorro`.

4.3.3 Algorithme de recouvrement de clef efficace pour les attaques différentielles

L'algorithme que nous présentons ici permet de tirer profit du faible nombre de boîtes-S d'un chiffrement PSPN pour réaliser un recouvrement de clef efficace. Plus précisément, nous montrons ici comment utiliser une caractéristique de probabilité p sur r tours pour faire une attaque sur $r + \lfloor 16/t \rfloor$ tours. En d'autres termes nous montrons comment retrouver la clef avec une attaque de complexité en temps proche de $2 \cdot p^{-1}$ bien que l'on laisse la différence se propager librement sur les $\lfloor 16/t \rfloor$ derniers tours du chiffrement.

Première intuition

Supposons que l'on souhaite réaliser une attaque différentielle tirant parti d'une caractéristique sur r tours de probabilité p pour attaquer $r+16/t$ tours du chiffrement. On note Δ_i, $i \in \{0, 1, \ldots, r\}$ la valeur de la différence définie par la caractéristique en fin de tour i, et on suppose pour simplifier notre explication que le nombre de boîtes-S par tour t divise 16.

La première étape de l'algorithme consiste à demander les chiffrés correspondant à p^{-1} paires de messages clairs de différence Δ_0, ce qui nous permet avec grande probabilité d'obtenir au minimum 1 paire qui suit la caractéristique, c'est-à-dire pour laquelle les différences intermédiaires sont effectivement Δ_i, $i \in \{0, 1, \ldots, r\}$.

Comme les chiffrés correspondent aux valeurs des messages après $r + 16/t$ tours du chiffrement (donc $16/t$ tours [16] après la fin de la caractéristique), il est très improbable que l'on possède une manière d'éliminer des paires de chiffrés invalides à partir de leur différence, et une attaque différentielle naïve serait donc mise en échec.

La question est alors de savoir comment obtenir de l'information à partir d'une paire de chiffrés et de la caractéristique.

L'algorithme que nous proposons fonctionne en deux étapes utilisant encore une fois intensivement la linéarisation :

— La première étape sert à déterminer pour chaque couple de chiffrés les valeurs des différences intermédiaires dans les $16/t$ derniers tours, donc en particulier la valeur des différences en entrée et en sortie de ses 16 boîtes-S. La définition de la boîte-S nous permet d'en déduire les valeurs possibles pour les états intermédiaires.

— La seconde étape nous permet de déduire les valeurs de clef possibles à partir des valeurs d'états renvoyées précédemment.

Chaque étape utilise une matrice pré-calculée (que l'on notera respectivement A_1 et A_2) et se réduit à très peu d'opérations linéaires. Nous les détaillons ci-après.

Première étape : Détermination des différences intermédiaires des $16/t$ derniers tours à partir de la différence de sortie

L'algorithme commence par calculer l'expression de la différence des chiffrés d'une bonne paire à partir de la différence (connue et fixée) Δ_r du dernier tour de la caractéristique. Ce calcul est réalisé en partant de Δ_r puis en suivant son évolution au cours des $16/t$ derniers tours : à chaque étape non linéaire, on introduit 8 nouvelles variables binaires pour représenter la différence en sortie de chacune des t boîtes-S. De cette façon, l'expression de la différence restera linéaire en la différence Δ_r et en les variables de linéarisation. Étant donné qu'on

16. Pour `Zorro` cela correspond à 4 tours.

introduit $8t$ nouvelles variables à chaque tour, la différence sur les chiffrés obtenue $\Delta_{r+16/t}$ s'exprimera comme une combinaison linéaire des bits de Δ_r et de $8t \cdot (16/t) = 128$ variables binaires.

Puisque ce calcul ne dépend aucunement de la valeur des chiffrés mais uniquement de la caractéristique, nous pouvons le réaliser dans la phase de pré-calcul. Pour minimiser plus avant le nombre d'opérations réalisées pour chaque paire de chiffrés, on mettra en forme le résultat de sorte qu'il exprime les transitions des boîtes-S en fonction de la différence des chiffrés.

Ceci est détaillé dans l'algorithme suivant, pour lequel on reprend la notation ST_i pour désigner la représentation symbolique de la différence au tour i, Δ_i. En particulier le premier état intervenant dans l'algorithme suivant est ST_r et est initialisé avec la différence en sortie de la caractéristique Δ_r.

Calcul de la matrice A_1 :

1. Pour chaque tour $i \in \{r, r+1, \ldots, r+16/t-1\}$:
   (a) Calculer ST_{i+1} à partir de ST_i en allouant $t \cdot 8$ nouvelles variables pour représenter la sortie des t boîtes-S du tour $i+1$ puis en calculant symboliquement l'étape linéaire L.
2. Partir de l'expression symbolique des 128 bits de $\Delta_{r+16/t}$ exprimés selon 128 variables de linéarisation et la modifier de sorte à obtenir la valeur de ces dernières en fonction de la différence des chiffrés. On note A_1 les 128 lignes de cette matrice donnant l'expression des différences de sortie des boîtes-S des tours [a] $r+1$ à $r+16/t$.

a. On pourrait se restreindre à $128 - 8t$ lignes correspondant aux différences en sorties des boîtes-S du tour $r+1$ à $r+16/t-1$ tours puisque les différences en sortie des dernières boîtes-S se déduisent directement de la différence des chiffrés.

L'étape précédente nous permet de calculer pour tout couple de chiffrés les valeurs des différences intermédiaires des $16/t$ derniers tours, et plus particulièrement d'obtenir les différences en entrée et en sortie de chacune des boîtes-S. On se réfère ensuite à la définition de celle-ci pour en déduire les valeurs [17] des octets d'entrée et de sortie puis, par extension, l'ensemble des valeurs des états intermédiaires sur ces tours. En supposant que la clef est introduite par l'opération `XOR` (comme pour la plupart des SPN) cette dernière information nous permet d'obtenir 128 équations linéaires en les bits de la (sous-)clef et donc de déterminer la clef [18].

Seconde étape : Détermination des différences intermédiaires des $16/t$ derniers tours à partir de la différence de sortie

L'algorithme suivant calcule la matrice A_2 (de 128 lignes et $128 + 256$ colonnes) permettant de calculer la (les) valeur(s) de (sous-)clefs candidates suggérées par un vecteur d'états intermédiaires donné.

17. Chaque transition en différence donnera en moyenne une valeur possible.
18. Si le cadencement de clef est réduit à l'identité ou est linéaire, on pourra retrouver la clef en résolvant un système linéaire. Dans le cas où le cadencement de clef est plus compliqué (comme pour l'AES par exemple), des techniques de type guess-and-determine seront plus efficaces.

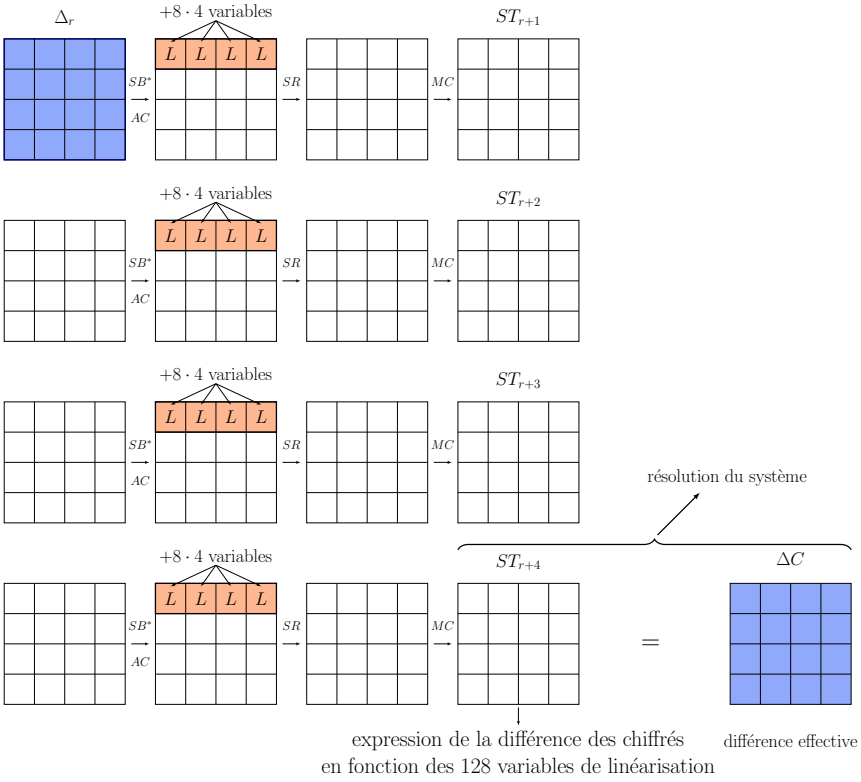


FIGURE 4.11 – Première étape du recouvrement de clef optimisé pour les PSPN (ici sur `Zorro`) : on détermine tout d'abord l'expression formelle de la différence des chiffrés en calculant les $16/t$ tours à partir de Δ_r et en linéarisant les boîtes-S. En exprimant l'égalité de cette expression avec la véritable valeur de ΔC, on retrouve l'ensemble des transitions des 16 boîtes-S.

Il consiste à exprimer les valeurs de sortie des 16 boîtes-S en fonction de :
— la valeur (symbolique) du chiffré C exprimée par 128 variables,
— les 128 variables représentant la (sous-)clef,
— les 128 variables de linéarisation introduites pour exprimer les valeurs d'entrée des boîtes-S.

Calcul de la matrice A_2 :

1. Allouer 128 variables pour représenter la valeur du chiffré C et initialiser un système vide S.
2. Inverser l'addition de sous-clef du tour $r + 16/t$: allouer 128 nouvelles variables pour représenter la valeur de la clef [a] et l'ajouter à C. On note cet état $ET_{r+16/t}$.
3. Inverser les $16/t$ derniers tours : pour tout $i \in \{r + 16/t, \cdots, r\}$:
   (a) Inverser l'étape linéaire en calculant $L^{-1}(ET_i)$.
   (b) Ajouter au système S l'expression de la sortie des boîtes-S obtenues.
   (c) Introduire 8 nouvelles variables pour chaque octet d'entrée des t boîtes-S du tour i.
4. Arranger le système par élimination gaussienne de façon à obtenir l'expression des 128 bits de la clef en fonction de la valeur des bits du chiffré, des $128 + 128$ variables d'entrée et de sortie des 16 boîtes-S. On note A_2 la matrice ainsi obtenue.

a. Cette étape est à adapter selon la fréquence d'insertion de la clef/des sous-clefs au cours des $16/t$ tours.

L'attaque complète est décrite (pour le cas $t = 4$) dans l'algorithme suivant :

Algorithme de recouvrement de clef optimal d'un PSPN :

1. Pré-calculer les matrices A_1 et A_2 (comme décrit précédemment).
2. Demander le chiffrement de p^{-1} paires de messages ayant comme différence en entrée Δ_0. Pour chaque paire (P, C) et(P', C'), faire :
   (a) Calculer la différence des chiffrés $\Delta_{r+4} = C \oplus C'$, et calculer $A_1 \cdot \Delta_{r+4}$ pour obtenir la valeur des octets de différences en entrée et en sortie des 16 boîtes-S des tours $r + 1$ à $r + 4$.
   (b) Se référer à la table des différences de la boîte-S pour déterminer si les transitions sont effectivement possibles. Si une ou plusieurs transitions sont impossibles, éliminer la paire et revenir au début de l'étape 2 pour analyser une autre paire.
   (c) En utilisant la définition de la boîte-S, déduire les valeurs possibles des paires d'octets en entrée et en sortie des boîtes-S. Regrouper dans une liste List l'ensemble des vecteurs des valeurs possibles pour les octets d'entrée et de sortie des 16 boîtes-S (donc des vecteurs de $8 \cdot (2 \cdot 16) = 256$ bits).
   (d) Pour chaque vecteur w de List, faire :
      i. Concaténer le chiffré de 128 bits C avec le vecteur de 256 bits w et noter v ce vecteur de 384 bits. Calculer $A_2 \cdot v$ pour obtenir une valeur candidate [a]

pour la (sous-)clef.

ii. Chiffrer un message avec la clef obtenue et le comparer au résultat donné par l'oracle de chiffrement. S'il y a égalité, retourner la clef.

a. On obtiendra en moyenne une clef candidate par paire.

Complexité finale de l'attaque

La complexité finale de l'attaque est la suivante :

— la complexité en données correspond aux p^{-1} paires de messages nécessaires pour espérer obtenir une paire suivant la caractéristique, donc $C_D = 2p^{-1}$.

— la complexité en temps se situe aux alentours de $4p^{-1}$ (correspondant aux $2p^{-1}$ chiffrements initiaux, suivis de la résolution de 2 systèmes d'équations linéaires pour chaque paire puis du test des clefs candidates (environ p^{-1}, qui peuvent être testées efficacement en s'assurant que la caractéristique est suivie tour après tour par exemple, ce qui évite de faire un chiffrement complet). On a donc $C_T \approx 4p^{-1}$.

— la complexité en mémoire est négligeable et se résume au stockage des 2 matrices A_1 et A_2 (la liste List peut être générée à la volée et n'a pas besoin d'être stockée).

Comme nous le montrerons lors de l'application de cette attaque à `Zorro`, il est possible d'augmenter le nombre de tours couverts par l'étape de recouvrement de clef si Δ_r possède des boîtes-S inactives et/ou impose que des boîtes-S des tours suivants sont inactives. En effet ceci permet de réduire le nombre de variables à introduire lors de la linéarisation et permet donc de couvrir plus de tours avant d'atteindre 128 nouvelles variables. Pour `Zorro` cela permet de couvrir un tour supplémentaire avec la même complexité (5 au lieu de $16/t = 4$) grâce à une caractéristique imposant que seulement 2 boîtes-S sont actives dans les tours $r+1$ et $r+2$.

4.3.4 Application des algorithmes précédents à `Zorro`

Dans cette section nous donnons les résultats de nos outils appliqués au chiffrement par blocs `Zorro`.

Notre premier algorithme nous permet de trouver les meilleures caractéristiques différentielles sur 19 tours de `Zorro` et plus précisément de prouver que les 6 caractéristiques itératives sur 4 tours trouvées indépendamment et au même moment par Rasoolzadeh et al. [RASA14] et par nous étaient les meilleures [19]. Leurs valeurs sont explicitées à la table 4.4.

A	00	A	00
B	E	B	E
C	00	C	00
D	F	D	F

	A	B	C	D	E	F
1	0x7b	0x88	0x23	0x83	0x55	0x2a
2	0xea	0x5c	0x5d	0xe4	0xa8	0x71
3	0xf7	0x16	0x8c	0x3a	0x4f	0xa8

TABLE 4.4 – Représentation générique et valeurs des 3 caractéristiques itératives sur 4 tours de probabilité $(6/256)^2$. Les 3 autres caractéristiques de même probabilité sont obtenues en échangeant les deux premières colonnes et les deux dernières colonnes, soit {*00, A, 00, A, E, B, E, B, 00, C, 00, C, F, D, F, D*}.

19. Leurs travaux n'avaient pas prouvé ce point et s'appuyaient sur une méthode mathématique utilisant la structure de `Zorro`.

L'application de notre second algorithme permet d'exploiter au mieux cette caractéristique pour monter une attaque sur les 24 tours (la version complète) de `Zorro`. Notre approche s'est montrée plus efficace que celle développée par Rasoolzadeh et al. puisque la complexité en temps de notre attaque est inférieure d'un facteur 2^{10} à la leur, bien que les deux attaques reposent sur la même caractéristique (voir table 4.3).

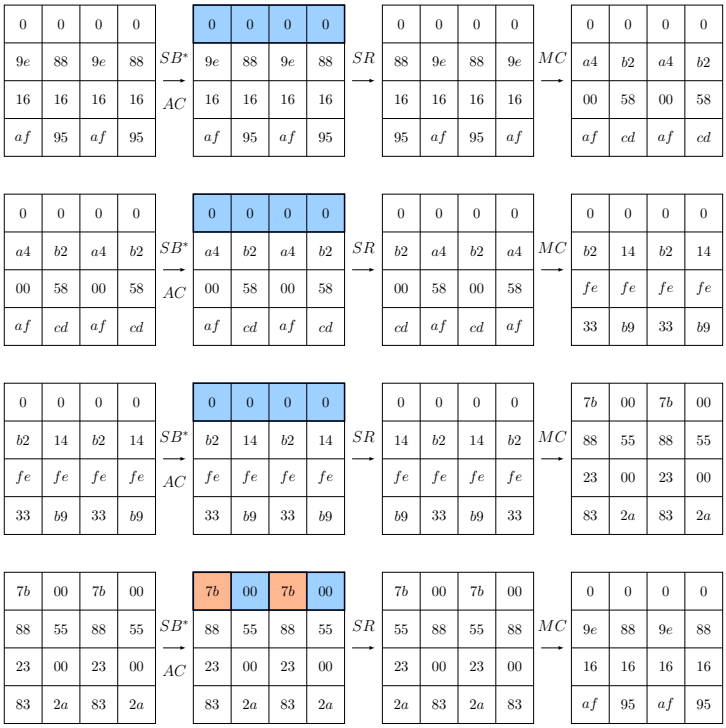


FIGURE 4.12 – Caractéristique itérative sur 4 tours de `Zorro`. La meilleure caractéristique sur 19 tours de `Zorro` est obtenue par l'itération de cette caractéristique (ou par l'itération des 5 autres décrites table 4.4).

Comme nous l'avons évoqué dans la section précédente, il est possible d'améliorer l'étape de recouvrement de clef par rapport à son exécution naïve : alors qu'une application directe nous permettrait de couvrir $16/4 = 4$ tours, on peut ici couvrir un tour de plus et attaquer la version complète avec une caractéristique sur 19 tours et un recouvrement de clef sur 5 tours.

Une des 6 meilleures caractéristiques sur 19 tours est représentée figure 4.13. Elle possède 8 boîtes-S actives et une probabilité de $(6/256)^8 \approx 2^{-43}$. Sa différence en sortie du tour 19 (donc la différence en entrée de l'opération SB^* du tour 20) est de la forme $\{A, 00, A, 00, B, E, B, E, C, 00, C, 00, D, F, D, F\}$ ce qui implique qu'uniquement 2 boîtes-S seront actives et devront être linéarisées au tour 20. Comme on le voit à la figure 4.13 à la page 80, 2 colonnes complètes de la caractéristique itérative sont maintenues en fin du tour 20 et cela quelles que soient les transitions suivies par les 2 boîtes-S actives de ce tour, ce qui implique que 2 boîtes-S seront inactives au tour 21. Au final il ne faudra linéariser que 4 boîtes-S aux tours 20 et 21, ce qui permet de linéariser 3 tours supplémentaires ensuite.

D'après l'analyse de complexité réalisée dans la section précédente, l'attaque finale aura

les complexités suivantes :

— $C_D = 2 \cdot p^{-1} = 2^{44}$ messages choisis,

— $C_T < 4 \cdot p^{-1} = 2^{45}$ opérations de chiffrements,

— C_M négligeable, moins de 2^{10} mots de 128 bits.

Nous pouvons améliorer plus avant la complexité en données de l'attaque en tirant parti du fait qu'il y a 6 caractéristiques de probabilité maximale sur 19 tours. Nous utilisons pour cela des structures de taille 2^6 : on part d'un message aléatoire quelconque auquel on ajoute toutes les 2^6 combinaisons linéaires possibles des 6 différences en entrée des 6 caractéristiques de probabilité 2^{-43}. À partir de ces 2^6 messages on peut former $2^5 \times 6$ paires ayant une différence en entrée parmi les 6 recherchées, à comparer aux 2^5 paires qu'on aurait pu former avec 2^6 messages si uniquement une différence était exploitable. Le gain en données est donc d'un facteur 6, réduisant à $2^{41.5}$ la complexité en données de l'attaque (on a une légère augmentation de la complexité en mémoire due à la nécessité de stocker les structures).

Pour corroborer notre analyse de complexité nous avons implémenté puis exécuté 11 fois la cryptanalyse de la version complète de `Zorro`. Les résultats obtenus sont donnés à la table 4.5. On peut y voir que la moyenne des simulations est très proche de la théorie. Ces résultats montrent que l'attaque fonctionne comme prévu.

Simulation	Messages chiffrés	Structures analysées	Paires analysées	Clefs suggérées
Théorie	$2^{41.5}$	$2^{35.5}$	2^{43}	2^{43}
Moyenne sur les simulations	$2^{41.49}$	$2^{35.49}$	$2^{43.07}$	$2^{43.07}$
1	$2^{38.30}$	$2^{32.30}$	$2^{39.89}$	$2^{39.84}$
2	$2^{38.50}$	$2^{32.50}$	$2^{40.08}$	$2^{40.06}$
3	$2^{38.56}$	$2^{32.56}$	$2^{40.14}$	$2^{40.10}$
4	$2^{38.77}$	$2^{32.77}$	$2^{40.35}$	$2^{40.34}$
5	$2^{38.86}$	$2^{32.86}$	$2^{40.44}$	$2^{40.44}$
6	$2^{39.12}$	$2^{33.12}$	$2^{40.70}$	$2^{40.69}$
7	$2^{40.59}$	$2^{34.59}$	$2^{42.17}$	$2^{42.22}$
8	$2^{40.83}$	$2^{34.83}$	$2^{42.41}$	$2^{42.43}$
9	$2^{42.90}$	$2^{36.90}$	$2^{44.49}$	$2^{44.47}$
10	$2^{43.07}$	$2^{37.07}$	$2^{44.66}$	$2^{44.67}$
11	$2^{43.21}$	$2^{37.21}$	$2^{44.79}$	$2^{44.79}$

TABLE 4.5 – Résultats des simulations de l'attaque différentielle sur la version complète de `Zorro`, comparés aux complexités théoriques. L'exécution la plus rapide a nécessité un peu moins de 8 heures, et la plus longue a nécessité environ 235.5 heures. Le programme a été implémenté en langage C et les résultats ont été obtenus avec des ordinateurs de bureaux.

4.4 Analyse des constructions PSPN basées sur l'AES

Dans cette section nous étudions les PSPN dont la construction est basée sur celle de l'AES et nous montrons que des caractéristiques linéaires et différentielles de forte probabilité du

type de celles qui ont été trouvées sur `Zorro` ont de grandes chances d'apparaître dans ces schémas.

Nous avons vu qu'il existe des caractéristiques de forte probabilité avec uniquement 2 boîtes-S actives tous les 4 tours pour `Zorro`. Comme cela avait été exposé dans l'attaque de Wang et al. [WWGY14], cette faiblesse provient de l'interaction entre les opérations *MixColumns* et *ShiftRows*. Nous étudions ici en détail ce phénomène.

Pour faciliter nos explications nous nous concentrons encore une fois sur le cas différentiel, le cas linéaire se traitant de la même manière.

Lorsqu'on s'intéresse aux caractéristiques ayant une forte probabilité pour `Zorro` (du type de celles données par Wang et al., ou de celles représentées figure 4.13 et table 4.4), on remarque qu'elles partagent les propriétés suivantes :

1. Elles sont itératives sur 4 tours
2. Leur différence dans les 2 premières colonnes est égale à leur différence dans les 2 dernières colonnes
3. Les transitions des boîtes-S actives sont de la forme $\Delta_{in} \rightarrow_S \Delta_{out} = \Delta_{in}$

La toute première attaque différentielle sur `Zorro` proposée par Wang et al. était un cas particulier des caractéristiques plus puissantes trouvées simultanément par Rasoolzadeh et al. [RASA14] et par nous. Elles possédaient un état totalement symétrique (4 colonnes identiques) ce qui leur permettait de *neutraliser* l'effet de l'opération *ShiftRows*. L'idée de ne considérer que des boîtes-S réalisant des transitions de typr $\Delta_{in} \rightarrow_S \Delta_{in}$ permettait en effet de *neutraliser* l'opération non-linéaire et d'obtenir une fonction de tour se résumant à l'opération MixColums, et d'utiliser le fait qu'elle soit d'ordre 4.

Pour trouver l'ensemble des caractéristiques satisfaisant les propriétés (1) et (3), on calcule les vecteurs propres de la matrice $(MC \circ SR)^4$ associés à la valeur propre 1. On obtient un espace propre de dimension 8 sur $\mathbb{F}_{2^8}$ correspondant aux états avec une symétrie dans les colonnes :

Propriété 4.1. *Les état Δ_0 satisfaisant l'équation $(MC \circ SR)^4(\Delta_0) = \Delta_0$ sont exactement tous les états pour lesquels les 2 premières colonnes sont égales aux 2 dernières colonnes.*

Cette proposition montre que les points (1) (les caractéristiques sont itératives) et (2) (symétrie des colonnes) sont équivalents.

Si on ne tient pas compte pour le moment de la spécification de la boîte-S, on a 2^{64} caractéristiques vérifiant (1), (2) et (3). Comme les opérations *MixColumns* et *ShiftRows* conservent toutes les deux la propriété de symétrie des 2 colonnes, les 2^{64} caractéristiques ont toutes leurs différences intermédiaires elles aussi symétriques sur 2 colonnes.

Étant donné que pour `Zorro` les boîtes-S sont positionnées de façon symétrique sur les 2 paires de colonnes la remarque précédente implique (sur 4 tours) que si k des 2×4 boîtes-S des colonnes de gauche sont inactives alors les k boîtes-S correspondantes dans les colonnes de droite seront inactives.

On peut facilement construire des caractéristiques itératives sur 4 tours possédant uniquement 2 boîtes-S actives en utilisant les remarques précédentes. On commence par sélectionner une des 2^{64} différences en entrée possibles ayant une symétrie sur 2 colonnes puis on écrit les équations correspondant à l'inactivité de 7 boîtes-S sur les 8 des colonnes de gauche. Ceci nous donne 56 équations linéaires en les bits de la différence en entrée. La résolution de ce

système nous donne 2^8 solutions (si les équations sont indépendantes) ayant 2 boîtes-S actives sur 4 tours.

En résumé, les particularités de la fonction de tour menant à ces caractéristiques de forte probabilité sont les suivantes :

1. Le grand nombre de points fixes de la fonction $(MC \circ SR)^4$.

2. Le fait que la propriété de symétrie par colonnes de ces points fixes soit conservée dans les tours intermédiaires.

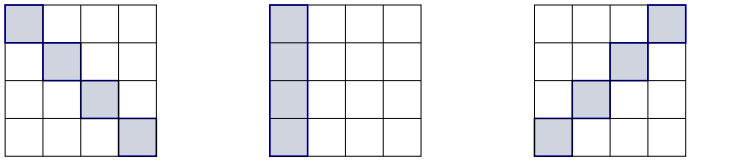
3. Le positionnement identique des boîtes-S dans ces 2 paires de colonnes.

Étant donné ces observations, plusieurs pistes sont envisageables pour améliorer la résistance aux attaques différentielles et linéaires d'un PSPN. Cependant, il faut garder à l'esprit que les changements doivent être en accord avec les critères de conception suivis par Gérard et al., à savoir que le PSPN doit être facile à masquer pour éviter les attaques par canaux auxiliaires et être léger. Ces critères nous incitent à conserver la même boîte-S (puisque celle-ci a été spécialement conçue pour atteindre un faible nombre d'opérations non-linéaires) ainsi qu'une opération *MixColumns* reposant sur une matrice circulante (ce qui limite la taille de l'implémentation).

La première piste que nous étudions ici est le changement de position des 4 boîtes-S de l'opération *SubBytes** de `Zorro`. Nous verrons ensuite l'effet du changement de matrice circulante puis finalement l'exemple d'un PSPN utilisant une version modifiée de *ShiftRows* associée à des boîtes-S positionnées différemment.

L'analyse donnée plus haut montre que changer le positionnement des boîtes-S de sorte qu'il ne soit plus symétrique sur 2 colonnes devrait permettre d'augmenter le nombre de boîtes-S actives. Rompre la symétrie permettrait en effet de ne plus avoir automatiquement l'annulation d'une boîte-S qui implique l'annulation d'une seconde boîte-S (sa symétrique dans l'autre paire de colonnes), et donc d'augmenter le nombre de boîtes-S actives total.

Nous avons donc étudié des variantes de `Zorro` possédant des boîtes-S dans des positions non symétriques. Les 3 cas que nous avons considérés sont ceux représentés à la figure 4.14.



(a) Première Diagonale (b) Première Colonne (c) Seconde Diagonale

FIGURE 4.14 – Autres positionnements des boîtes-S étudiés.

	Différentielle	Linéaire
(a) Première Diagonale	2^{-32}	$2^{-11.96}$
(b) Première Colonne	2^{-33}	$2^{-11.96}$
(c) Seconde Diagonale	2^{-32}	$2^{-14.16}$

TABLE 4.6 – Probabilités maximales des caractéristiques itératives sur 1 *étape* (4 tours) des 3 variantes de `Zorro`.

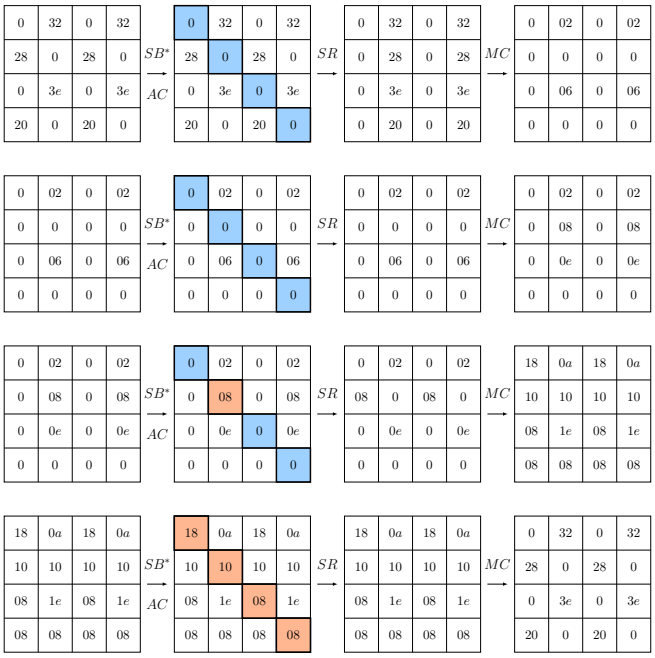


FIGURE 4.15 – Exemple de meilleure caractéristique itérative sur 4 tours de la variante de `Zorro` avec des boîtes-S sur la première diagonale (probabilité de 2^{-32}).

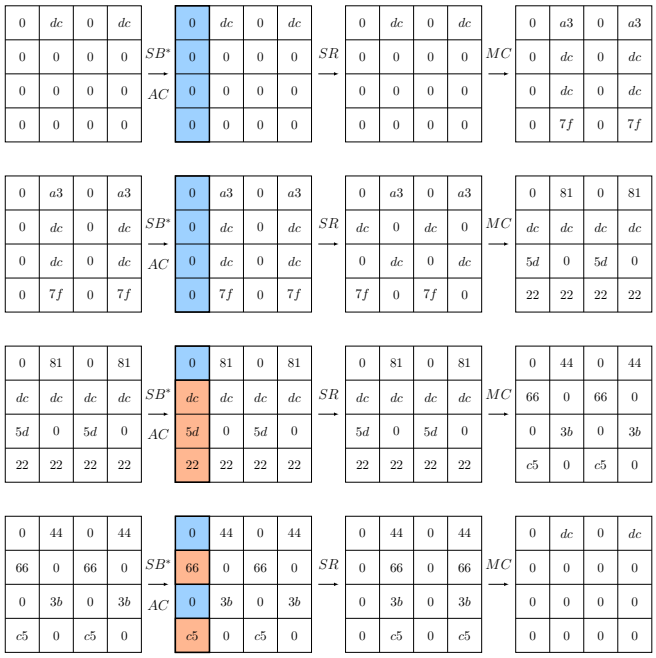


FIGURE 4.16 – Exemple de meilleure caractéristique itérative sur 4 tours de la variante de `Zorro` avec des boîtes-S sur la colonne (probabilité de 2^{-33}). Bien que l'on ait aussi 5 boîtes-S actives comme pour la première diagonale, les probabilités des transitions sont plus faibles d'où la probabilité de 2^{-33} au lieu de 2^{-32}.

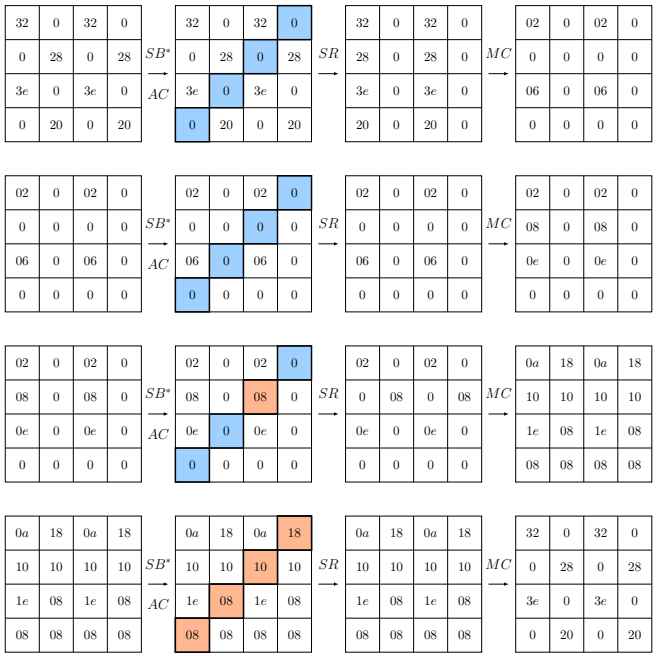


FIGURE 4.17 – Exemple de meilleure caractéristique itérative sur 4 tours de la variante de `Zorro` avec des boîtes-S sur la seconde diagonale (probabilité de 2^{-32}).

Comme prévu, changer uniquement le positionnement des boîtes-S ne permet pas d'éliminer l'existence de caractéristiques itératives mais permet ici de diminuer leur probabilité. Comme nous allons le montrer dans la section suivante ces 3 variantes n'atteignent pas les propriétés que l'on pourrait espérer d'un chiffrement PSPN *idéal*[20] et c'est pourquoi nous étudions un autre variante pour laquelle l'opération *ShiftRows* est, elle aussi, différente.

4.5 Construction d'un PSPN résistant aux attaques différentielles

Objectifs

Nous établissons dans cette section que la faiblesse de `Zorro` face aux attaques linéaires et différentielles n'est pas intrinsèque à la construction PSPN — c'est-à-dire à la décision de diminuer le nombre de boîtes-S par tour — mais provient d'un choix malheureux de ses composants. En effet, nous avons été capables de construire une version légèrement modifiée de `Zorro` qui résiste aux attaques auxquelles celui-ci est vulnérable.

Pour quantifier les propriétés que l'on peut attendre d'un PSPN, considérons un PSPN idéal avec t boîtes-S par tour et pour lequel l'étape linéaire de chaque tour est choisie aléatoirement dans l'espace des fonctions linéaires inversibles (on peut raisonnablement supposer que la diffusion obtenue est très rapide).

Comme nous l'avons vu dans la section 4.3.1, une caractéristique avec a boîtes-S actives sur r tours donne lieu à un système de $8(t \cdot r - a)$ équations linéaires en $8(16 + a)$ variables. En s'appuyant sur l'hypothèse que les étapes linéaires sont aléatoires, on peut s'attendre à ce qu'il existe une solution si $8(16 + a) \geq 8(t \cdot r - a)$, c'est-à-dire si $a \geq (tr - 16)/2$. Ceci implique qu'un tel PSPN aura au minimum $(tr - 16)/2$ boîtes-S actives sur r tours[21].

Cette borne doit être manipulée prudemment puisqu'elle simplifie à l'extrême la situation : il faut garder à l'esprit qu'il existe de nombreux motifs possibles et qu'il suffit qu'une caractéristique soit valide pour pouvoir monter une attaque différentielle. Un autre aspect plaide dans l'autre sens : pour s'assurer qu'une caractéristique est valide il ne suffit pas de s'assurer qu'il existe des solutions au système linéaire mais il faut aussi s'assurer que les transitions des boîtes-S définies par celui-ci sont possibles, ce qui pour chaque boîte-S correspond à une probabilité d'environ 1/2.

Cependant, on peut raisonnablement considérer que ces deux effets se compensent et qu'au final la borne $a \geq (tr - 16)/2$ est un objectif raisonnable pour qu'un PSPN soit considéré comme *'bon'* (remarquons ici que, selon ce nouveau critère, les 3 variantes décrites à la section précédentes ne sont pas bonnes : en répétant les caractéristiques itératives on peut par exemple obtenir seulement 21 boîtes-S actives sur 19 tours alors que $(tr-16)/2 = (4 \cdot 19-16)/2 = 30$).

Si on considère l'AES qui est un cas extrême du (P)SPN avec $t = 16$ et qu'on applique cette formule pour obtenir une estimation du nombre minimal raisonnable de boîtes-S actives que l'on peut espérer sur $r = 4$ tours, on obtient qu'une caractéristique sur 4 tours devrait avoir un minimum de $a \geq (16 \cdot 4 - 16)/2 = 24$ boîtes-S actives. Il est facile de démontrer que ce minimum vaut en réalité 25, ce qui montre que notre estimation est raisonnable dans ce cas.

20. De façon générale l'existence de caractéristiques itératives n'est jamais souhaité.
21. On peut voir cette condition comme le souhait que les équations obtenues soient indépendantes et donc que le calcul des degrés de liberté soit correct.

Analyse de notre construction

Nous construisons ici un PSPN qui satisfait notre formule $a \geq (tr-16)/2$ pour des grandes valeurs de r et résiste donc mieux aux attaques différentielles et linéaires de base que `Zorro`. Si on se réfère à la discussion précédente cela impose que notre construction s'éloigne de celle de l'AES pour éviter les caractéristiques itératives sur 4 tours. La faiblesse des constructions de type AES provient de la combinaison des deux propriétés suivantes :

— Toute matrice MDS M élevée à la puissance 4 possède un grand nombre de vecteurs propres satisfaisant $M^4(x) = \alpha x$ pour une valeur de α fixée.

— L'ordre de *ShiftRows* est 4 (c'est-à-dire que *ShiftRows* appliqué 4 fois est l'identité).

Ainsi, pour éviter les caractéristiques de forte probabilité existant pour les schémas de type AES nous devons construire un PSPN s'éloignant de cette construction en nous assurant qu'au minimum une des deux propriétés ci-dessus n'est pas satisfaite.

La construction que nous proposons considère une version modifiée de *ShiftRows* possédant un ordre supérieur à 4 et qui diffère également par le positionnement des 4 boîtes-S. Ce dernier changement permet de contrer l'attaque publiée par Yanfeng Wang et al. [WWGY14] (rappelée en début de chapitre) qui s'avère être insensible à la définition du *ShiftRows* puisqu'elle utilise une caractéristique à 4 colonnes identiques.

La variante de *ShiftRows* que nous avons choisie est représentée à la figure 4.18 et fonctionne de la façon suivante : la transformation subie par les lignes 1, 3 et 4 sont les mêmes que dans le *ShiftRows* original de l'AES, mais le traitement réalisé sur la seconde ligne est modifié de sorte à augmenter l'ordre de la transformation. L'octet numéro 8 reste inchangé, alors que les 3 autres sont décalés de 2 positions vers la gauche. L'ordre de la transformation de cette ligne est 3, celle de la première ligne est 1, celle de la ligne 3 est 2 et celle de la dernière ligne est 4. Cela implique que l'opération SR^* ainsi définie est d'ordre $3 \cdot 4 = 12$.

Étant donné l'immobilité de l'octet 8 cette variante de *ShiftRows* apporte moins de diffusion locale que la version originale. Cependant, son ordre est supérieur et nous allons voir qu'au final elle renforce la résistance de la construction aux attaques différentielles et linéaires de base.

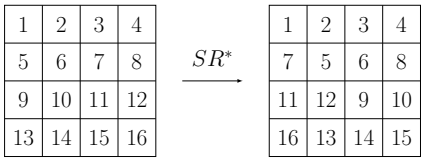


FIGURE 4.18 – Version modifiée de *ShiftRows* considéré dans notre variante de `Zorro`.

Si on se réfère à la discussion précédente, cette variante s'approche du PSPN idéal si son nombre a de boîtes-S actives sur r tours vérifie $a \geq (4 \cdot r - 16)/2$, ce qui implique qu'on espère $a \geq (4 \cdot 11 - 16)/2 = 14$ boîtes-S actives sur 11 tours.

Cependant, il ne semble pas totalement invraisemblable qu'une caractéristique avec 13 boîtes-S actives soit possible : le nombre de motifs à analyser est $\binom{11 \cdot 4}{13} > 2^{35}$, chacun déterminant un système linéaire à $8 \cdot (16 + 13) = 8 \cdot 29$ variables et $8 \cdot (44 - 13) = 8 \cdot 31$ équations, donnant lieu à une solution avec probabilité $2^{8 \cdot (29-31)} = 2^{-16}$. Obtenir quelques caractéristiques valides (avant vérification des transitions) avec 13 boîtes-S actives sur 11 tours ne semble donc pas totalement impossible.

Nous avons appliqué notre outil de recherche de caractéristiques à cette variante de `Zorro` réduite à 11 tours ce qui nous a permis de montrer qu'il n'existe pas de caractéristique différentielle avec au plus 12 boîtes-S actives (quelle que soit la boîte-S). Si on relance la recherche avec $a = 13$, on trouve quelques dizaines de caractéristiques (avant filtrage), mais aucune n'est valide lorsqu'on considère les transitions possibles pour la boîte-S de `Zorro`. Notre borne semble donc valide pour cette variante.

Nous avons pu montrer que le nombre minimum de boîtes-S actives d'une caractéristique sur 11 tours est de 14. Notre outil trouve environ 2^{32} caractéristiques (avant filtrage), et environ 2^{22} sont valides lorsqu'on considère la boîte-S de `Zorro`. La meilleure caractéristique sur 11 tours de notre nouvelle construction est représentée à la figure 4.19. Sa probabilité est d'environ $2^{-86.8}$.

Comme l'uniformité différentielle de la boîte-S de `Zorro` est 10 ($\approx 2^{-4.7}$) nous pouvons en déduire que la meilleure caractéristique sur 11 tours a une probabilité de $\min(2^{-86.8}, 2^{-4.7 \cdot 15}) = 2^{-70.5}$. Par conséquent, la meilleure caractéristique différentielle sur 22 tours aura une probabilité inférieure à $(2^{-70.5})^2 = 2^{-141}$ (cette borne est meilleure (pour le concepteur) si on considère la boîte-S de l'AES d'uniformité différentielle 4).

Pour 12 tours du chiffrement nous avons pu montrer qu'il n'existe pas de caractéristique différentielle avec au plus 14 boîtes-S actives. La recherche pour des valeurs supérieures de a devenant trop longue pour un ordinateur de bureau standard, nous n'avons pas calculé le nombre minimum de boîtes-S actives pour 12 tours. Cependant, les résultats que nous avons pu obtenir sont suffisants pour démontrer que la résistance aux attaques différentielles de base de cette version est supérieure à celle de `Zorro`, et s'approche de ce qu'on pourrait attendre d'un PSPN idéal.

Une caractéristique de probabilité au plus 2^{-141} correspond (avec la boîte-S de `Zorro`) à une caractéristique d'au plus $a = 30$ boîtes-S actives, ce qui avec notre formule $a \geq (tr-16)/2$ correspond à $r = 19$ tours. Ainsi, l'écart entre le PSPN idéal et ce que nous avons pu prouver avec notre outil est de seulement 3 tours.

Nous ne nous risquons pas ici à proposer une spécification complète de cette variante de `Zorro`. Nous nous sommes assurés que cette variante est résistante aux attaques différentielles et linéaires *de base* mais pour que notre proposition soit sérieuse il faudrait s'assurer qu'elle résiste aux variantes d'attaques différentielles (tronquées, impossibles, ...) et linéaires (zéro-corrélation, multidimensionnelles, ...) mais aussi aux attaques d'autres types. Ces analyses nous permettraient notamment de déterminer le nombre de tours recommandés pour cette variante.

4.6 Conclusion

L'apport des recherches sur la construction proposée par Benoît Gérard et ses coauteurs que nous avons présentées dans ce chapitre est multiple :

— Tout d'abord, nous avons résolu une des questions laissées en suspens par les concepteurs de `Zorro`, à savoir trouver une technique permettant d'analyser la sécurité d'un chiffrement PSPN contre les attaques différentielles et linéaires de base. Nous avons en effet conçu plusieurs algorithmes permettant de déterminer les meilleures caractéristiques existant pour un nombre de tours fixé. Il faut ensuite comparer ces caractéristiques à ce que nous avons déterminé comme étant les propriétés souhaitables pour ce type de construction.

— Ces algorithmes nous ont permis d'étudier la sécurité de l'instance particulière de PSPN proposée dans [GGNS13], appelée `Zorro`, et de monter la meilleure attaque différentielle connue sur sa version complète. La complexité de notre attaque fait que celle-ci est réalisable en pratique et que nous avons pu l'implémenter pour vérifier sa validité.
— Nos recherches ont réussi à élucider les raisons de l'existence de caractéristiques de forte probabilité, à savoir l'interaction entre les opérations *MixColumns* et *ShiftRows*, toutes deux d'ordre 4.
— Comme nos algorithmes sont génériques, nous avons pu les appliquer à plusieurs PSPN et décrire une variante de `Zorro` qui résiste aux attaques différentielles et linéaires de base et s'approche de ce que nous avons défini comme le comportement idéal d'un PSPN. Nous avons pour cela suivi la direction suggérée par le point précédent, à savoir s'éloigner d'une construction type AES. Notre exemple prouve que la construction proposée par Gérard et ses coauteurs n'est pas intrinsèquement faible mais pourrait être réutilisée — en s'assurant tout de même de sa résistance aux autres types d'attaques — dans le futur.

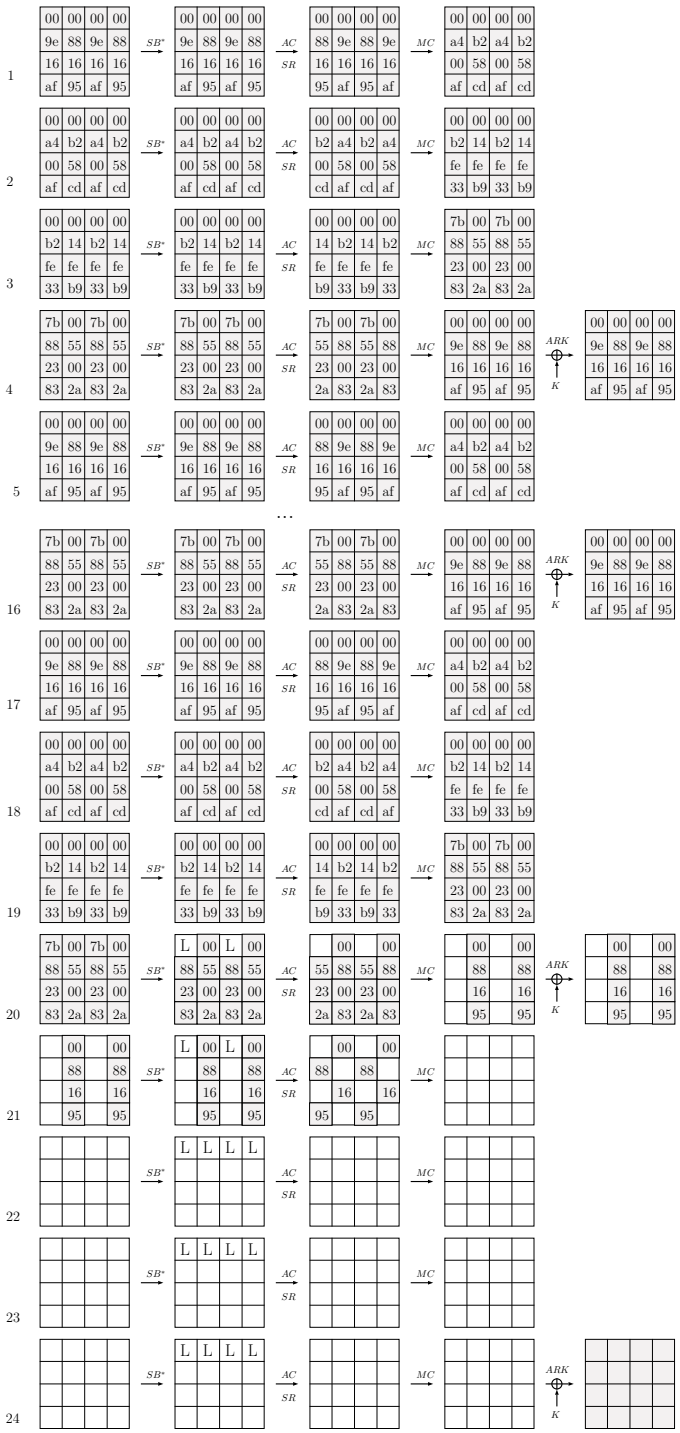


FIGURE 4.13 – Attaque différentielle sur la version complète de `Zorro` : la différence en sortie des 16 boîtes-S notées d'un L sont initialement inconnues : elles sont linéarisées puis déterminées lors de l'étape de recouvrement de clef comme expliqué à la section 4.3.3.

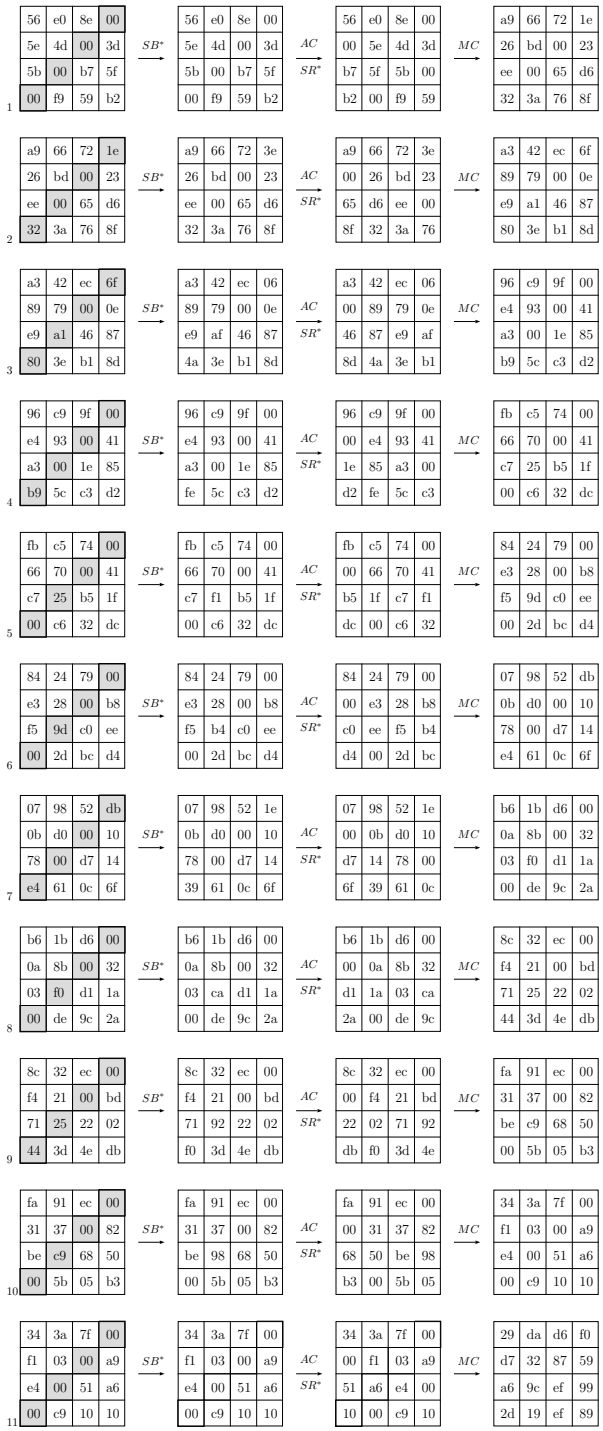


FIGURE 4.19 – Meilleure caractéristique sur 11 tours (nombre de boîtes-S actives minimum de 14) sur la version modifiée de `Zorro`.

Chapitre 5

Attaques à clefs liées sur le chiffrement par blocs PICARO

Dans ce chapitre, nous nous intéressons au chiffrement par blocs PICARO [PRC12a], conçu par Gilles Piret, Thomas Roche et Claude Carlet en 2012. L'analyse de ce chiffrement, réalisée avec Anne Canteaut et María Naya-Plasencia, a abouti à une attaque à clefs liées de la version complète qui fut présentée à la conférence SAC 2015 [CLN15].

5.1 Description du chiffrement par blocs PICARO et remarques préliminaires

5.1.1 Description de PICARO

Comme `Zorro` (paru l'année suivante), le chiffrement par blocs PICARO a été conçu pour proposer une alternative à l'AES offrant de meilleures performances lors de la mise en place de contre-mesures aux attaques par canaux auxiliaires. Sa taille de bloc et sa sécurité sont toutes les deux de 128 bits.

Comme exposé au chapitre 4, la contre-mesure aux attaques physiques la plus populaire est le schéma de masquage, qui trouve sa source dans les techniques de partage de secret. Le schéma considéré ici par Piret, Roche et Carlet est le masquage booléen de Rivain et Prouff [RP10]. Lors de sa mise en place, celui-ci induit un surcoût proportionnel à l'ordre de masquage pour les opérations linéaires, mais quadratique pour les multiplications (c'est ce que nous avons détaillé à la section 4.1.3).

Cet aspect orienta l'ensemble de la conception de PICARO, à commencer bien entendu par le choix de sa boîte-S, et amena aussi les auteurs à préférer un cadencement de clef entièrement linéaire. Nous détaillons ces choix de conception dans les paragraphes suivants.

Boîte-S

Les parties les plus complexes à gérer lors de la mise en place du schéma de masquage étant les opérations non-linéaires, Piret et ses coauteurs débutèrent leurs recherches par l'étude de la boîte-S.

Pour que celle-ci soit facile à masquer, elle devait avoir une description simple sous forme d'un polynôme creux traduisant un nombre restreint de multiplications dans le corps fini.

Le véritable défi ici était de trouver une boîte-S facile à masquer tout en ayant aussi de bonnes propriétés vis-à-vis des attaques dites logiques (cryptanalyse linéaire, différentielle, algébrique...).

Leur analyse les conduisit à la sélection d'une boîte-S non bijective de 8 bits analysée par Claude Carlet dans un article paru dans le journal *Designs, Codes and Cryptography* en 2011 [Car11]. Cette boîte-S est formée par la concaténation de deux polynômes bivariés : le premier prend en argument deux variables x et y de $\mathbb{F}_{2^4}$ et retourne leur produit, tandis que le second retourne le produit des cubes des deux variables d'entrée translatées d'une constante :

$$\begin{array}{rcl} S : \mathbb{F}_{2^4} \times \mathbb{F}_{2^4} & \longrightarrow & \mathbb{F}_{2^4} \times \mathbb{F}_{2^4} \\ (x, y) & \longmapsto & (xy, (x^3 + \texttt{0x02})(y^3 + \texttt{0x04})) \end{array} \tag{5.1}$$

où `0x02` et `0x04` sont des notations hexadécimales représentant des éléments du corps $\mathbb{F}_{2^4}$ défini par $\mathbb{F}_2[X]/(X^4 + X^3 + 1)$.

Cette boîte-S a de très bonnes propriétés cryptographiques puisqu'elle est différentiellement 4-uniforme (c'est-à-dire qu'elle a une probabilité différentielle maximale de 2^{-6}) et possède une non-linéarité de 94 ainsi qu'un degré algébrique de 4.

Son point fort à l'égard du schéma de masquage est qu'elle s'exprime avec seulement 4 multiplications, qui plus est dans le petit corps $\mathbb{F}_{2^4}$: le premier polynôme nécessite une multiplication $(x \times y)$ alors que le second polynôme demande une élévation au carré et une multiplication pour chaque élévation au cube, 2 additions puis une multiplication ($((x)^2 \times x + \texttt{0x02}) \times ((\texttt{y})^2 \times \texttt{y} + \texttt{0x04})$), soit au total 4 multiplications (non linéaires), 2 additions et deux élévations au carré pour toute la boîte-S. Comme vu au chapitre 4, protéger une addition à l'ordre d nécessite $d + 1$ additions (une pour chacune des $d + 1$ parts), de même une élévation au carré nécessite $d + 1$ élévations au carré dans la version masquée. Chaque multiplication demande :

— $(d + 1)^2$ multiplications,
— $2d(d + 1)$ additions,
— $d(d + 1)/2$ nombres aléatoires de 4 bits.

Le nombre d'opérations nécessaires au calcul de la version masquée à l'ordre d de la boîte-S de PICARO est de :

— $4(d + 1)^2$ multiplications,
— $4(2d(d + 1)) + 2(d + 1) = (8d + 2)(d + 1)$ additions,
— $2(d + 1)$ mises au carré,
— $2d(d + 1)$ nombres aléatoires de 4 bits.

On rappelle que la boîte-S de l'AES [1] peut, elle aussi, s'exprimer avec uniquement 4 opérations non linéaires, mais celles-ci sont réalisées dans $\mathbb{F}_{2^8}$, ce qui la rend plus difficile à masquer [2].

La table des valeurs de la boîte-S de PICARO est donnée à la table 5.1.

1. La comparaison des caractéristiques des boîtes-S de PICARO, `Zorro` et de l'AES est donnée à la table 4.1 du chapitre 4.

2. Ce point est expliqué dans [PRC12a] par le fait qu'il est possible de mettre en table la multiplication dans le corps $\mathbb{F}_{2^4}$ alors que cela serait trop volumineux avec $\mathbb{F}_{2^8}$.

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	08	0c	03	06	06	04	05	06	05	04	0c	0c	04	03	05	03
10	0a	1f	29	3b	4b	55	62	7b	82	95	af	bf	c5	d9	e2	f9
20	01	2d	45	6a	8a	ac	cf	ea	9f	bc	dd	fd	1c	35	5f	75
30	0f	34	61	52	c2	fb	a3	92	13	2b	74	44	db	e1	b3	81
40	0f	44	81	c2	92	db	13	52	b3	fb	34	74	2b	61	a3	e1
50	0e	59	a4	f8	d8	87	7c	28	3c	67	99	c9	e7	b4	4c	14
60	02	63	ca	ad	1d	71	d7	bd	27	41	e3	83	31	5a	f7	9a
70	0f	74	e1	92	52	2b	b3	c2	a3	db	44	34	fb	81	13	61
80	02	83	9a	1d	bd	31	27	ad	f7	71	63	e3	41	ca	d7	5a
90	0e	99	b4	28	f8	67	4c	d8	7c	e7	c9	59	87	14	3c	a4
a0	0a	af	d9	7b	3b	95	e2	4b	62	c5	bf	1f	55	f9	82	29
b0	0a	bf	f9	4b	7b	c5	82	3b	e2	55	1f	af	95	29	62	d9
c0	0e	c9	14	d8	28	e7	3c	f8	4c	87	59	99	67	a4	7c	b4
d0	01	dd	35	ea	6a	bc	5f	8a	cf	1c	fd	2d	ac	75	9f	45
e0	02	e3	5a	bd	ad	41	f7	1d	d7	31	83	63	71	9a	27	ca
f0	01	fd	75	8a	ea	1c	9f	6a	5f	ac	2d	dd	bc	45	cf	35

TABLE 5.1 – Boîte-S de PICARO. Les 4 bits de poids fort de l'entrée correspondent à la variable x utilisée dans la définition 5.1 de S, tandis que les 4 bits de poids fort correspondent à y.

Quelques remarques sur la distribution de cette boîte-S sont données en annexe B.

Fonction de tour

Étant donné que cette boîte-S n'est pas bijective [3], la façon la plus naturelle de l'intégrer dans un chiffrement est d'utiliser un schéma de Feistel. Cependant, comme discuté par les auteurs dans la spécification du chiffrement [PRC12a], l'intégration naïve d'une boîte-S non bijective dans un schéma de Feistel peut mener à une attaque différentielle dévastatrice n'activant qu'une boîte-S tous les 2 tours, comme représenté à la figure 5.1. En effet, on pourrait dans certains cas trouver une différence δ sur les messages qui n'active qu'une seule boîte-S au tour 2 et qui soit annulée en sortie de cette boîte-S avec grande probabilité. Dans ce cas-ci, la différence en entrée du tour suivant est nulle et la situation est identique à celle du premier tour. Si on note p la probabilité d'annulation de la différence en sortie de la boîte-S et R le nombre de tours du chiffrement, on obtient ainsi une caractéristique itérative dont la probabilité est $p^{\lfloor \frac{R}{2} \rfloor}$. Une attaque linéaire analogue est aussi possible.

Pour contrer ce type d'attaque, les concepteurs ont introduit une fonction d'expansion Exp et une fonction de compression Comp permettant d'assurer un nombre minimum de boîtes-S actives dans une caractéristique (différentielle ou linéaire). La structure de la fonction de tour résultant de leur recherche est représentée à la figure 5.2. La fonction de tour opère sur la branche de gauche (de 64 bits) et est composé de 4 opérations :

3. Il est beaucoup plus facile de construire une boîte-S possédant de bonnes propriétés cryptographiques dans ce cas.

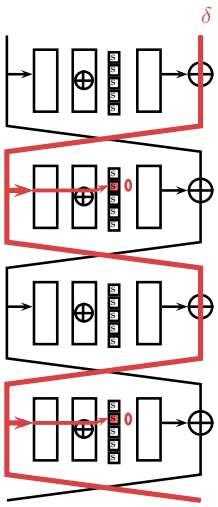


FIGURE 5.1 – Exemple d'attaque possible sur les schémas de Feistel utilisant des boîtes-S non bijectives : on peut choisir une différence en entrée de sorte qu'une seule boîte-S soit active (et annule la différence) un tour sur deux.

— **Expansion** (Exp) : l'entrée de la fonction de tour est étendue linéairement en un état de 112 bits,
— **Addition de clef** (ARK) : la clef de tour k^i lui est additionnée,
— **Étage de boîtes-S** (SB) : chacun des 14 octets est transformé par la boîte-S,
— **Compression** (Comp) : l'état obtenu est compressé à nouveau en 64 bits.

La fonction d'expansion Exp est construite à partir d'un code MDS de paramètres [14, 8, 7] déduit d'un code de Reed-Solomon raccourci. La propriété MDS permet d'assurer que, si l'entrée de la fonction d'expansion est active, alors sa sortie aura un minimum de 7 octets actifs ce qui permet d'éviter l'attaque différentielle précédente. La matrice définissant l'expansion G est la suivante :

$$G = \begin{pmatrix} 01 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 01 & 01 & 0A & 01 & 09 & 0C \\ 00 & 01 & 00 & 00 & 00 & 00 & 00 & 00 & 05 & 01 & 01 & 0A & 01 & 09 \\ 00 & 00 & 01 & 00 & 00 & 00 & 00 & 00 & 06 & 05 & 01 & 01 & 0A & 01 \\ 00 & 00 & 00 & 01 & 00 & 00 & 00 & 00 & 0C & 06 & 05 & 01 & 01 & 0A \\ 00 & 00 & 00 & 00 & 01 & 00 & 00 & 00 & 09 & 0C & 06 & 05 & 01 & 01 \\ 00 & 00 & 00 & 00 & 00 & 01 & 00 & 00 & 01 & 09 & 0C & 06 & 05 & 01 \\ 00 & 00 & 00 & 00 & 00 & 00 & 01 & 00 & 0A & 01 & 09 & 0C & 06 & 05 \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 01 & 01 & 0A & 01 & 09 & 0C & 06 \end{pmatrix}$$

Chaque élément de la matrice est défini dans le corps $\mathbb{F}_2[X]/(1 + X^2 + X^3 + X^4 + X^8)$.

La fonction de compression transforme une entrée de taille 112 bits en 64 bits et est définie par la matrice $H = G^T$. L'utilisation de la transposée de G permet d'assurer la résistance de PICARO à l'équivalent linéaire de l'attaque différentielle de la figure 5.1 puisqu'elle assure qu'un masque linéaire non nul impactera un minimum de 7 octets.

La fonction de tour est itérée 12 fois.

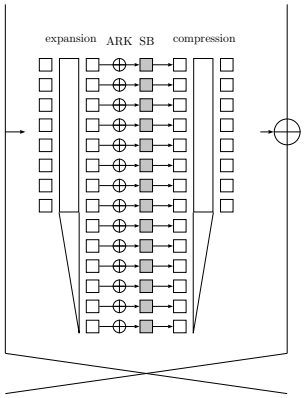


FIGURE 5.2 – Fonction de tour de PICARO : la moitié gauche de l'état interne est étendue de 64 à 112 bits puis est additionnée à la clef de tour avant de passer à travers l'étage de boîtes-S. Elle est finalement recompressée à sa taille initiale puis additionnée à la moitié droite de l'état interne.

Algorithme de cadencement de clef

La fonction de chiffrement nécessite 12 clefs de tour de 112 bits chacune. Les concepteurs de PICARO souhaitaient un algorithme de cadencement de clef qui soit résistant aux deux attaques les plus communes sur les clefs de tours que sont les attaques à clefs liées (*related-key attacks*) [Bih94] et les attaques par glissement (*slide attacks*) [BW99], tout en étant simple à implémenter. Cette dernière contrainte se traduit par la restriction à des opérations linéaires de base telles que les rotations et les sélections de bits. Une autre propriété visée était la possibilité de calculer les clefs de tour à la volée, c'est-à-dire de pouvoir déduire la clef $i+1$ à partir de la clef i (en chiffrement) ainsi que la clef $i-1$ à partir de la clef i (lors du déchiffrement).

La définition de l'algorithme de cadencement de clef faisant apparaître cette propriété est donnée ci-après. Elle consiste à d'abord calculer une clef étendue formée de 12 éléments de de 128 bits chacun notés κ^i ($i \in [1,12]$) puis à ne conserver que les 112 premiers bits de chaque κ^i pour former les clefs de tour k^1 à k^{12}.

$$\begin{cases} \kappa^1 = K \\ \kappa^i = T(\kappa^{i-1}) \ggg \theta(i) \qquad i \in [1,12] \end{cases}$$

où θ est défini par :

i	2	3	4	5	6	7	8	9	10	11	12
$\theta(i)$	1	15	1	15	1	52	1	15	1	15	1

et où T est la fonction linéaire définie par :

$$\begin{pmatrix} T(K)^{(1)} \\ T(K)^{(2)} \\ T(K)^{(3)} \\ T(K)^{(4)} \end{pmatrix} = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \times \begin{pmatrix} K^{(1)} \\ K^{(2)} \\ K^{(3)} \\ K^{(4)} \end{pmatrix}$$

La fonction T étant involutive, une définition équivalente de l'algorithme de cadencement de clef est :

$$\begin{cases} \kappa^1 = K \\ \kappa^i = T(K) \ggg \Theta(i) & \text{pour } i = 2, 4, 6, 8, 10, 12 \\ \kappa^i = K \ggg \Theta(i) & \text{pour } i = 3, 5, 7, 9, 11 \end{cases}$$

où $\Theta(i)$ vaut :

i	2	3	4	5	6	7	8	9	10	11	12
$\Theta(i)$	1	16	17	32	33	85	86	101	102	117	118

Des formules équivalentes existent pour le calcul des clefs en déchiffrement. Nous renvoyons à la spécification [PRC12b] pour leur définition.

5.1.2 Remarques préliminaires

Notre attaque repose sur deux observations très simples : la première concerne la boîte-S, la seconde porte sur la structure de la fonction de tour.

1. **Boîte-S de PICARO.** En construisant la table de distribution des différences de la boîte-S de PICARO, on peut remarquer que toutes les différences d'entrée peuvent être annulées :

   Propriété 5.1. *Soit $\delta \in \mathbb{F}_{2^8}$ la différence d'entrée de la boîte-S de PICARO. On a :*

   $$\forall \delta \in \mathbb{F}_{2^8},\ \exists\ x \in \mathbb{F}_{2^8} \mid S(x) \oplus S(x \oplus \delta) = 0.$$

   De plus, pour tout δ non nul, on a exactement 2 valeurs solutions :

   $$\forall \delta \in \mathbb{F}_{2^8}^*,\ P(\delta \to_S 0) = 2^{-7}.$$

2. **Structure de la fonction de tour.** Par simple observation de la fonction de tour, on peut remarquer que l'addition de clef se situe après l'expansion (donc une différence introduite dans la clef ne sera pas diffusée par celle-ci) et se situe avant la couche de boîtes-S, ce qui permet d'annuler immédiatement une différence dans les clefs.

Ces deux remarques nous suggèrent d'envisager une situation telle que celle représentée à la figure 5.3 : on va considérer un scénario d'attaque à clefs liées et plus précisément demander le chiffré de messages sous deux clefs-maîtres liées entre elles par une différence conduisant à très peu de différences dans les clefs de tour. On souhaite pouvoir construire un distingueur basé sur l'annulation immédiate de ces différences par passage des boîtes-S.

5.2 Construction d'un distingueur à clefs liées sur la version complète de PICARO

5.2.1 Description de l'objectif visé

Nous examinons ici la question de savoir s'il est possible de construire une différence entre deux clefs-maître telle que *avec grande probabilité*[4] un même message clair M chiffré sous ces

4. C'est-à-dire avec probabilité supérieure à la probabilité générique valant 2^{-128}.

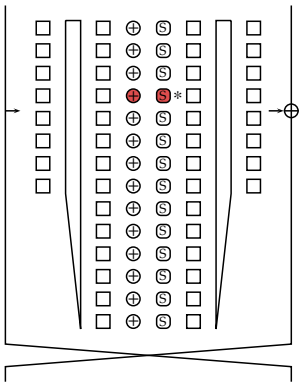


FIGURE 5.3 – Représentation de l'idée de base de notre attaque : une différence introduite dans les clefs de tours (en rouge sur le dessin) peut être immédiatement annulée en passant dans la boîte-S.

deux clefs aboutisse aux mêmes chiffrés :

$$\exists ?\ \Delta \in \mathbb{F}_2^{128} \mid \text{pour un couple aléatoire } (M,K) \quad P(E_K(M) = E_{K\oplus\Delta}(M)) > 2^{-128}.$$

Si on souhaite utiliser l'idée précédente et annuler une différence aussitôt qu'elle apparaît en la faisant passer par la boîte-S, étant donné que chaque annulation a une probabilité de 2^{-7}, on pourra *désactiver* un maximum de s boîtes-S, où s vérifie :

$$2^{-7s} > 2^{-128},$$

ce qui implique que s devra être inférieur à 18.

Il est assez facile de construire une différence sur la clef-maître qui n'active que 18 boîtes-S, simplement en observant la structure de l'algorithme de cadencement de clef[5], mais ici nous souhaitons trouver *toutes* les différences de clef-maître activant un *minimum* d'octets dans les sous-clefs.

5.2.2 Interprétation du problème en une recherche de mots de petits poids d'un code linéaire

La méthodologie que nous avons suivie se base sur l'observation que l'algorithme de cadencement de clef est totalement linéaire sur $\mathbb{F}_2$. L'ensemble des suites de sous-clefs produites peut donc être assimilé à un code linéaire, ce qui permet de ramener notre problème à un problème de recherche de mots de plus petit poids d'un code. Cependant, il faut noter que le code est linéaire sur $\mathbb{F}_2$, mais que la quantité qui nous intéresse ici est le poids en terme d'octets (correspondant au nombre de boîtes-S actives).

Plus précisément, le code que nous considérons contient tous les mots formés par la concaténation des sous-clefs de tour (de 112 bits) k^i, $i \in [1,12]$; il est donc de dimension $k = 128$ (la taille de la clef-maître) et de longueur $n = 112 \times 12 = 1344$. Sa matrice génératrice est représentée à la figure 5.4.

5. En plaçant judicieusement 2 différences dans la clef-maître à 32 bits d'écart par exemple, ce qui permet de limiter la propagation des différence lors de l'application de la fonction T.

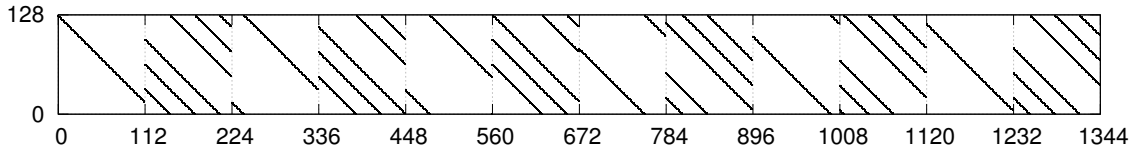


FIGURE 5.4 – Représentation graphique de la matrice génératrice du code linéaire défini par l'algorithme de cadencement de clef.

Ce schéma donne une assez bonne idée de l'algorithme de cadencement de clef : on peut y voir que les sous-clefs impaires (indices $[224i, 224i + 112[$ pour $i \in [0, 5]$) sont une simple sélection des bits de la clef-maître, et que les sous-clefs paires ont, elles aussi, une structure similaire entre elles (provenant de la fonction T).

Pour résoudre notre problème de recherche de mot de plus petit poids, une première idée serait d'utiliser un des nombreux algorithmes dédiés : en effet ce problème est central en théorie des codes et de nombreuses solutions y ont été apportées, parmi lesquelles l'algorithme de Canteaut-Chabaud [CC98].

Cependant, étant donné la très grande longueur du code et sa taille relativement petite, nous allons voir qu'il est très facile de trouver ces mots par une recherche exhaustive limitée à des mots de poids 4 dans la clef-maître.

Remarque 5.1. *Un travail réalisé récemment par Petr Lisonek et Layla Trummer [LT16] décrit un nouvel algorithme de recherche de distance minimale particulièrement adapté aux matrices ayant une structure similaire à celle de G. Cet algorithme, nommé matroid partitioning algorithm, permet de calculer en moins de 2 minutes la distance minimale du code défini par G ainsi que de retrouver les 8 mots de poids minimum que nous donnons dans la table 5.2. En comparaison, et comme rapporté dans [LT16], le logiciel* MAGMA *[BCP97] estime le temps nécessaire pour compléter le calcul de la distance minimale de G par sa fonction* `MinimumWeight` *à 10^6 années.*

Le fait que nous puissions réaliser la recherche de mots de petits poids en parcourant uniquement les mots de poids 4 dans la clef-maître provient de la définition de l'algorithme de cadencement de clef. L'observation que nous faisons est la suivante : l'algorithme de cadencement de clef définit tous les κ^{2i+1}, $i \in [0, 5]$ par une simple sélection des bits de clef-maître : ainsi, si un bit est à 1 dans K, il se retrouvera dans les 6 κ impairs. Les sous-clefs k^{2i+1}, $i \in [0, 5]$ étant formées par les 112 premiers bits des κ correspondants, certains des bits à 1 peuvent se retrouver dans la partie tronquée. Une rapide recherche montre qu'uniquement 2 bits à 1 peuvent être éliminés de cette façon. Ainsi, tout bit à 1 dans la clef-maître produira un minimum de 4 bit à 1 dans les sous-clefs d'indice impair. Par conséquent, si on cherche à obtenir un mot de poids inférieur ou égal à 18, on sait qu'on peut limiter notre recherche aux clefs-maîtres de poids inférieur ou égal à 4. Cette recherche indique que le poids minimum est 18, et retourne les 8 mots consignés dans la table 5.2.

On peut remarquer que ces configurations sont assez prévisibles : elles correspondent à des différences dans la clef-maître positionnées à 32 bits d'écart, de telle sorte que, lors du passage par la fonction T comme par les rotations, chaque sous-clef a un maximum de 2 bits à 1, qui plus est positionnés de façon à apparaître le plus souvent possible dans les parties tronquées.

n°	K	k^1	k^2	k^3	k^4	k^5	k^6	k^7	k^8	k^9	k^{10}	k^{11}	k^{12}
1	27, 123	27	28	11, 43	12, 44	27, 59	28, 60	80	81	0, 96	1, 97	16	17
2	28, 124	28	29	12, 44	13, 45	28, 60	29, 61	81	82	1, 97	2, 98	17	18
3	29, 125	29	30	13, 45	14, 46	29, 61	30, 62	82	83	2, 98	3, 99	18	19
4	30, 126	30	31	14, 46	15, 47	30, 62	31, 63	83	84	3, 99	4, 100	19	20
5	91, 123	91	92	11, 107	12, 108	27	28	48, 80	49, 81	64, 96	65, 97	80	81
6	92, 124	92	93	12, 108	13, 109	28	29	49, 81	50, 82	65, 97	66, 98	81	82
7	93, 125	93	94	13, 109	14, 110	29	30	50, 82	51, 83	66, 98	67, 99	82	83
8	94, 126	94	95	14, 110	15, 111	30	31	51, 83	52, 84	67, 99	68, 100	83	84

TABLE 5.2 – Description des mots (représentant les sous-clefs) atteignant un poids minimum égal à 18.

5.2.3 Résultats

Les configurations reportées à la table 5.2 ne correspondent pas exactement à ce qui nous intéresse. En effet, on souhaite déterminer les mots de plus petit poids en terme d'octets non nuls et non en terme de bits puisque nous nous intéressons au nombre minimum de boîtes-S actives à *annuler*. Cependant, les résultats de la recherche précédente réalisée dans $\mathbb{F}_2$ montrent que l'on peut atteindre 18 octets actifs avec 2 octets actifs dans la clef-maître : pour obtenir le résultat voulu en terme d'octets on se restreint donc à tous les mots possédant un maximum de 2 octets actifs dans la clef-maître. Une recherche exhaustive sur ce type de configurations montre qu'il existe 30 différences de ce type, formées par les $(2^4 - 1) = 15$ combinaisons non nulles des 4 premières configurations et des $(2^4 - 1) = 15$ combinaisons non nulles des 4 dernières configurations de la table 5.2. On note W cet ensemble de différences sur les clefs-maîtres.

L'ensemble de cette analyse montre qu'il existe un (faible) biais dans le chiffrement PICARO, qui aurait dû se comporter comme une permutation aléatoire mais qui vérifie la propriété suivante :

Propriété 5.2. *La probabilité que les chiffrés correspondant à un message aléatoire chiffré sous deux clefs liées par une différence appartenant à l'ensemble W soient identiques est 2^{-126}.*

Cette propriété est représentée à la figure 5.5 : on choisit une différence de clef-maître dans W et on souhaite que la différence de chaque sous-clef s'annule immédiatement par passage dans la boîte-S, ce qui arrive avec probabilité 2^{-7} pour chaque boîte-S.

5.3 Utilisation du distingueur pour monter une attaque à clefs liées

5.3.1 Problématique

Dans cette section nous cherchons à étendre le distingueur précédent en une attaque à clefs liées. Le scénario considéré ici est celui où l'attaquant peut demander le chiffrement de messages sous la clef secrète et sous une clef liée à celle-ci par une différence qu'il choisit.

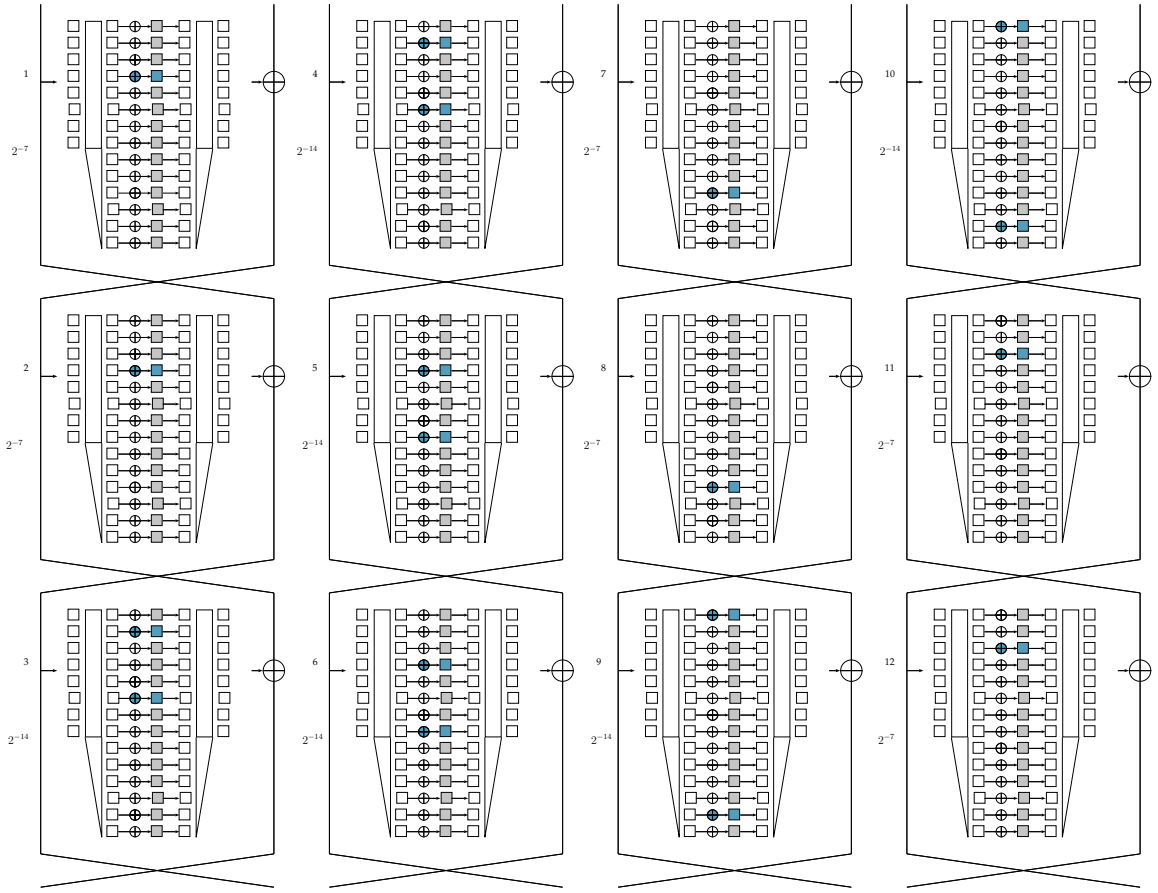


FIGURE 5.5 – Représentation du distingueur : lorsqu'on choisit une différence dans W (ici la première configuration décrite table 5.2), une collision sur les chiffrés arrive avec probabilité 2^{-126} dans le cas de PICARO, alors que dans le cas d'un chiffrement idéal cette probabilité est 4 fois moins fréquente (2^{-128}).

Une première idée possible pour adapter le distingueur précédent en une attaque à clefs liées pourrait être de réaliser une *nR-attaque* similaire aux attaques différentielles de base : placer un distingueur sur les $12 - n$ premiers tours, puis inverser les n derniers tours avec une clef candidate, et déduire que la bonne clef est celle faisant apparaître la propriété du distingueur.

Le problème ici est que si on souhaite utiliser un distingueur basé sur l'annulation de toutes les différences par les boîte-S, la probabilité finale d'un tel événement sera très faible ce qui rend l'obtention de plusieurs paires qui suivent la différentielle très coûteuse voire impossible. Nous allons donc plutôt utiliser une attaque assez similaire à celle développée pour KLEIN au chapitre 3, c'est-à-dire utiliser uniquement une paire se conformant à la différentielle, et nous servir des tours précédents pour éliminer les mauvaises hypothèses de clef.

L'attaque que nous décrivons ici utilise un chemin différentiel couvrant les 10 premiers tours (en imposant que chacune des boîtes-S activée par la clef a une différence en sortie nulle) et considère une étape de recouvrement de clef sur 2 tours. On notera dans la suite a_i, $i \in [1, 12]$ le nombre de boîtes-S actives à chaque tour et $a_{i \to j}$ le nombre de boîtes-S actives

du tour i au tour j.

5.3.2 Construction d'une étape de recouvrement de clef sur 2 tours

Les deux propriétés nous permettant de faire fonctionner notre attaque alors que la différence se propage librement sur les 2 derniers tours sont les suivantes :

1. Si une paire suit la caractéristique différentielle, seulement une petite fraction des différences sur les chiffrés est atteignable,
2. La fonction de compression est inversible à faible coût.

Nous les détaillons et donnons leurs preuves ci-dessous. On considère pour le moment [6] que les paires de messages considérées sont formées de textes clairs identiques, chiffrés sous deux clefs liées.

Différences atteignables par les chiffrés si la caractéristique est suivie

Propriété 5.3. *Si une paire de messages correspondant au même texte clair chiffré par deux clefs liées suit la caractéristique différentielle sur* 10 *tours alors le nombre maximal de différences atteignables sur la branche droite des chiffrés est de* $2^{7a_{11}}$.

Démonstration. Nous utilisons la figure 5.6 pour illustrer notre raisonnement. Comme expliqué précédemment, nous considérons une situation pour laquelle les 10 premiers tours sont couverts par une caractéristique similaire à celle utilisée dans le distingueur précédent, c'est-à-dire pour laquelle on impose que toutes les différences des clefs soient annulées en traversant les boîtes-S, alors que la différence se propage librement sur les 2 derniers tours du chiffrement.

Si un même message, chiffré avec des clefs liées, suit cette caractéristique, cela implique que la différence en entrée du tour 11 est nulle pour chaque branche ($\delta L_{10} = 0$ et $\delta R_{10} = 0$) mais aussi que la différence δR_{12} (qui est la différence de la moitié droite du chiffré) est égale à la différence de gauche en sortie de la fonction de tour du tour 11. Comme au tour 11 aucune condition n'est imposée sur les a_{11} différences présentes dans la sous-clef, on a (au maximum) $2^{7a_{11}}$ différences possibles après passage des boîtes-S. La fonction de compression étant linéaire, on retrouve ce même nombre de possibilités en sortie de la fonction de tour. L'attaquant peut très facilement déterminer cet ensemble de différences possibles par lecture de la table de distribution des différences de la boîte-S puis par calcul de la fonction linéaire de compression, et ainsi en déduire l'ensemble restreint des différences possibles pour le chiffré droit d'une paire suivant la caractéristique.

Cette propriété permet d'éliminer facilement des paires qui ne suivent pas la caractéristique : l'attaquant devra simplement comparer la différence des chiffrés des paires qu'il possède avec celles de l'ensemble qu'il a pré-calculé. La probabilité qu'une paire aléatoire passe ce test est de $2^{-64+7a_{11}}$.

□

6. Nous montrerons dans la section 5.4 comment optimiser notre attaque en considérant des clairs différents.

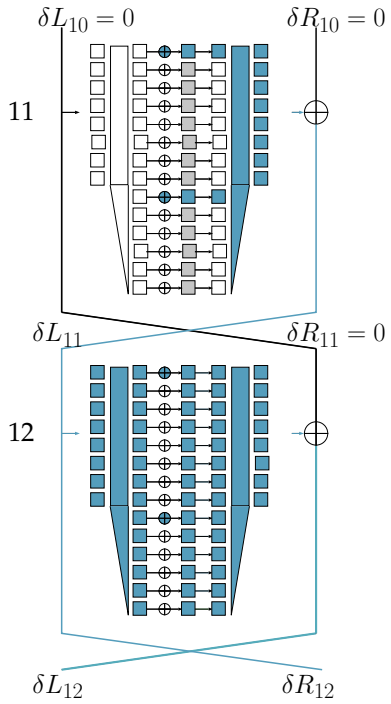


FIGURE 5.6 – Si une paire suit la caractéristique différentielle, seulement une petite fraction des différences sur les chiffrés est atteignable.

Propriété de la fonction de compression

La propriété que nous donnons ici permet de montrer que la connaissance de la valeur de l'état en sortie de la fonction de compression ainsi que de 6 octets de l'entrée (sur les 14) permet de déterminer de façon unique l'intégralité de son entrée. Comme la fonction de compression est linéaire, cette propriété est aussi valide pour les différences.

Propriété 5.4. *Deux mots distincts de 14 octets ayant la même image par la fonction de compression coïncident sur au maximum 5 octets.*

Démonstration. H est définie par $H = G^T$, où G est la matrice génératrice d'un code MDS. En conséquent, H correspond à une matrice de contrôle de parité d'un code MDS, et elle est elle-même la matrice génératrice d'un code MDS de distance minimale $14 - 6 + 1 = 9$.

Ainsi, si deux entrées distinctes de la compression ont la même sortie et coïncident sur 6 octets de l'entrée, on obtient une contradiction puisque cela signifierait qu'elles différent sur un maximum de $14 - 6 = 8$ octets alors que la distance minimale est de 9. □

Comme annoncé précédemment, cette proposition implique que la connaissance de la valeur des 8 octets en sortie de Comp et de 6 de ses octets d'entrée permet de calculer la valeur de l'ensemble des 14 octets d'entrée. Elle interviendra dans notre attaque pour calculer la valeur de la différence en entrée de la fonction de compression du dernier tour. Pour cela, nous nous basons sur l'observation suivante : si la paire de messages considérée suit la caractéristique différentielle, alors la différence observée dans la moitié gauche du chiffré correspondra à la différence en sortie de la dernière fonction de compression (voir figure 5.7). L'attaquant peut alors faire une hypothèse sur 6 octets de la différence en entrée de Comp (en

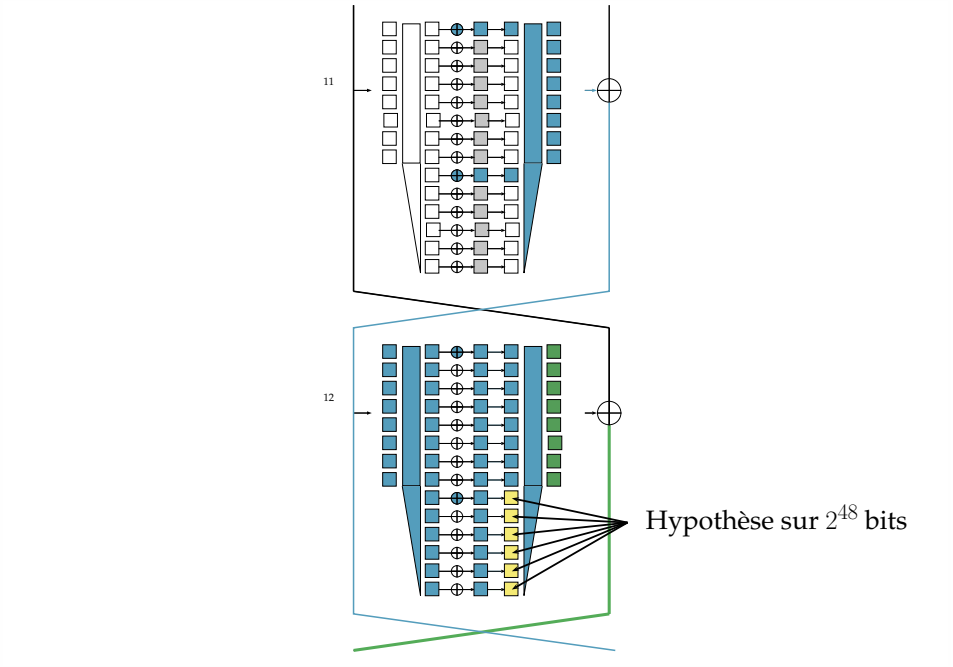


FIGURE 5.7 – Utilisation de la propriété d'inversion de la fonction de compression dans l'attaque : la différence en sortie étant connue, on réalise une hypothèse sur 6 octets de la différence en entrée de laquelle on déduit les 8 octets restants.

jaune sur la figure 5.7) et ainsi déterminer de façon unique la valeur associée des 14 octets de différence en entrée de Comp. Bien que ce procédé soit répété de nombreuses fois dans notre attaque, son coût restera raisonnable puisqu'il ne fait intervenir que de l'algèbre élémentaire et des opérations linéaires.

5.3.3 Description générique de l'attaque

Grâce à ces deux propriétés nous pouvons décrire une première version de l'attaque :

1. Choisir une différence de clef-maître Δ minimisant le nombre d'octets actifs dans les sous-clefs du tour 1 à 10 ($a_{1\to 10}$) et demander le chiffrement de $2^{7\times a_{1\to 10}}$ messages aléatoires M_i chiffrés avec la clef secrète K ainsi qu'avec la clef liée $K \oplus \Delta$. On notera (M_i, C_i, C'_i) le triplet associé à chaque message.
2. Utiliser la propriété 5.3 sur les différences de chiffrés possibles pour éliminer des triplets qui, de façon sûre, ne suivent pas la différentielle. Il reste $2^{7\times a_{1\to 10}-64+7a_{11}}$ triplets après cette opération de filtrage.
3. Utiliser la propriété 5.4 de la fonction de compression pour déterminer les 112 bits de la différence en entrée de la fonction de compression du tour 12. Comme cela nécessite de deviner 6 octets de différence, on obtient 2^{48} possibilités différentes pour chaque triplet, soit au total $2^{7\times a_{1\to 10}-64+7a_{11}+48} = 2^{7\times a_{1\to 11}-16}$ candidats, chacun formé d'un triplet (M_i, C_i, C'_i) associé à 6 octets d'hypothèse.
4. Comme cela apparaît sur la figure 5.6, la différence observée dans la partie droite du chiffré correspond à la différence en entrée de la fonction de tour 12. On en déduit

la valeur de la différence en sortie de la fonction d'expansion et par conséquent la différence après l'étape d'addition de clef (en effet la différence dans les sous-clefs est connue). À cette étape on possède alors la différence en entrée et en sortie de la couche de boîtes-S.

5. En utilisant la table de distribution des différences de S, vérifier si les transitions obtenues sont valides pour chacune des 14 boîtes-S de la couche non-linéaire du tour 12. Éliminer toutes les configurations invalides. Dans le cas où toutes les transitions sont bien de probabilité strictement positive, calculer les valeurs possibles des états intermédiaires en entrée de la couche de boîtes-S. Comme, pour chaque boîte-S, le produit entre la probabilité qu'une transition soit valide et le nombre de valeurs permettant cette transition est égale à 1, le nombre d'états intermédiaires retournés par cette procédure est égal au nombre de candidats à l'étape précédente, soit $2^{7\times a_{1\to 11}-16}$ candidats.
6. Calculer la valeur de l'état intermédiaire en entrée de l'étape d'addition de clef en évaluant la fonction d'expansion sur la valeur de la moitié droite du chiffré.
7. Additionner cette valeur à celle en entrée de la couche de boîtes-S calculée précédemment pour en déduire une valeur candidate pour la sous-clef du tour 12, k^{12}. On obtient un total de $2^{7\times a_{1\to 11}-16}$ candidats pour k^{12}.

Cette procédure demande la connaissance de $2\times 2^{7\times a_{1\to 10}}$ paires clairs-chiffrés pour obtenir les $2^{7\times a_{1\to 10}}$ triplets nécessaires à l'attaque. Les étapes 6 et 7 au cours desquelles les valeurs possibles pour k^{12} sont calculées nécessitent $2^{7\times a_{1\to 11}-16}$ chiffrements sur 1 tour de la paire de messages, soit $2^{7\times a_{1\to 11}-16}\times 2\times \frac{1}{12} = 2^{7\times a_{1\to 11}-16}\times 2^{-18.58}$ chiffrements complets.

Les autres étapes sont de complexité négligeable en comparaison, donc la complexité en temps du calcul des valeurs candidates pour k^{12} est de $2^{7\times a_{1\to 10}+1} + 2^{7\times a_{1\to 11}-18.58}$.

Les candidats possibles pour k^{12} peuvent être testés en réalisant environ $2^{7\times a_{1\to 11}-16+16} = 2^{7\times a_{1\to 11}}$ chiffrements complets, correspondant à rechercher exhaustivement les 16 bits restant pour obtenir K à partir de la valeur candidate pour k^{12} et à exécuter un chiffrement-test. La complexité totale de l'attaque atteindrait alors la valeur de : $2^{7\times a_{1\to 10}+1} + 2^{7\times a_{1\to 11}}$ (le coût des étapes 6 et 7 devient négligeable face à celui des chiffrements-tests).

5.4 Optimisations et compromis possibles

5.4.1 Deux optimisations

Optimisation 1 : La première optimisation que nous décrivons ici sert à diminuer le second terme apparaissant dans l'expression de la complexité en temps de l'attaque. Elle est obtenue en remarquant qu'il est possible de tester plus simplement si une clef candidate est valide : on va pour cela vérifier graduellement, tour après tour, si la caractéristique est suivie et éliminer un candidat dès lors qu'une transition de boîte-S est invalide. On commencera par vérifier ceci sur les tours les plus accessibles (le premier et l'avant-dernier), et à l'intérieur de ceux-ci on commencera par vérifier les transitions des boîtes-S pour lesquelles il faut réaliser le moins d'hypothèses de clef possible.

Une configuration d'autant plus favorable à cette méthode est celle où la valeur de k^{12} permet de déterminer sans hypothèse supplémentaire la valeur des bits de clef de k^1 et k^{11}

nécessaires au calcul des valeurs en entrée des boîtes-S actives des tours 1 et 11. Ces tours-là sont en effet directement accessibles et cette information nous permet à moindre coût d'éliminer une clef candidate fausse.

Une recherche des meilleures configurations retourne les résultats présentés aux tables 5.3 et 5.4.

Avec cette technique, le test des clefs candidates repasse à un coût inférieur à celui des étapes 6 et 7 de la procédure décrite à la section précédente, ce qui implique qu'on a :

$$C_T = 2^{7\times a_{1\to 10}+1} + 2^{7\times a_{1\to 11}-18.58}$$

Optimisation 2 : Le nombre de boîtes-S à annuler étant assez élevé, la probabilité de la caractéristique est assez faible, et par conséquent le nombre de paires à générer avant d'en obtenir une qui soit valide est élevé. Pour réduire un peu le coût de la génération des données, nous proposons ici d'utiliser une technique similaire à celle des structures (utilisées pour les attaques différentielles) que nous avons notamment utilisée dans l'attaque sur KLEIN (voir chapitre 3). Son principe est le suivant : au lieu de chiffrer le même message avec deux clefs différentes et de considérer une caractéristique pour laquelle toutes les différences sont immédiatement annulées, on laisse la différence de la première sous-clef de tour se propager librement et on cherche à l'annuler par une différence provenant de la moitié droite du message (voir figure 5.8).

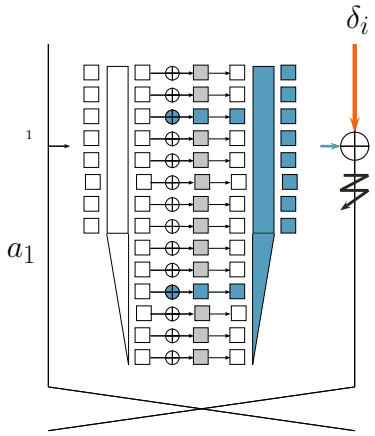


FIGURE 5.8 – Optimisation 2 : la différence de la sous-clef du tour 1 se propage librement dans la fonction de tour et est annulée par introduction d'une différence dans la moitié droite des messages clairs. Cela permet de former des structures et de diminuer le nombre de chiffrés nécessaires à l'attaque.

Avec ce principe, si on veut construire une paire qui suit la caractéristique du premier tour, il faut que la différence des messages dans la moitié droite corresponde à la différence en sortie de la fonction de compression (provenant de la différence de la sous-clef du tour 1). Comme la valeur de cette différence n'est pas connue de l'attaquant, une possibilité est de demander le chiffrement de tous les 2^{8a_1} messages $P \oplus \delta$ où δ couvre toutes les différences possibles à la sortie de la fonction de compression. Avec cette méthode, l'attaquant est assuré d'avoir 1 paire conforme à la caractéristique du premier tour, mais cela nécessite un total de $2^{8a_1} + 1$ chiffrements (voir figure 5.9). Une autre méthode, préférable, consiste à chiffrer ces

positions des différences		a_1	$a_{2\to10}$	a_{11}	$a_{1\to11}$	octets connus (bits) k_1	octets connus (bits) k_{11}
11	107	2	14	2	18	1 (14)	2 (16)
12	108	2	14	2	18	1 (14)	2 (16)
13	109	2	14	2	18	1 (14)	2 (16)
14	110	2	14	2	18	1 (14)	2 (16)
15	111	2	14	2	18	1 (14)	2 (16)
16	80	2	14	2	18	2 (16)	2 (16)
17	81	2	14	2	18	2 (16)	2 (16)
18	82	2	14	2	18	2 (16)	2 (16)
19	83	2	14	2	18	2 (16)	0 (14)
20	84	2	14	2	18	2 (16)	0 (14)
21	85	2	14	2	18	2 (16)	0 (14)
22	86	2	14	2	18	2 (16)	0 (14)
23	87	2	14	2	18	2 (16)	0 (14)
24	88	2	14	2	18	0 (4)	0 (14)
25	89	2	14	2	18	0 (4)	0 (14)
26	90	2	14	2	18	0 (4)	0 (14)
27	91	2	14	2	18	0 (4)	0 (0)
28	92	2	14	2	18	0 (4)	0 (0)
29	93	2	14	2	18	0 (4)	0 (0)
30	94	2	14	2	18	0 (4)	0 (0)
31	95	2	14	2	18	0 (4)	0 (0)
32	96	2	14	2	18	0 (0)	0 (0)
33	97	2	14	2	18	0 (0)	0 (0)
34	98	2	14	2	18	0 (0)	0 (0)
35	99	2	14	2	18	0 (0)	0 (2)
36	100	2	14	2	18	0 (0)	0 (2)
37	101	2	14	2	18	0 (0)	0 (2)
38	102	2	14	2	18	0 (0)	0 (2)
39	103	2	14	2	18	0 (0)	0 (2)
40	104	2	14	2	18	0 (12)	0 (2)
41	105	2	14	2	18	0 (12)	0 (2)

TABLE 5.3 – Résultats retournés par une recherche exhaustive des différences de clef-maître (avec moins de 4 bits de différence) minimisant $a_{2\to10}$ ($a_{2\to10} = 14$) et ayant une valeur de $a_{1\to11}$ minimale ($a_{1\to11} = 18$). La dernière colonne indique le nombre d'octets (de bits) de clef connus aux positions des boîtes S actives au tour 1 et 11 si k^{12} est connue.

positions des différences		a_1	$a_{2\to10}$	a_{11}	$a_{1\to11}$	octets connus (bits) k_1	octets connus (bits) k_{11}
27	123	1	15	1	17	0(2)	0(0)
28	124	1	15	1	17	0(2)	0(0)
29	125	1	15	1	17	0(2)	0(0)
30	126	1	15	1	17	0(2)	0(0)
91	123	1	15	1	17	0(2)	0(0)
92	124	1	15	1	17	0(2)	0(0)
93	125	1	15	1	17	0(2)	0(0)
94	126	1	15	1	17	0(2)	0(0)

TABLE 5.4 – Résultats retournés par une recherche exhaustive des différences de clef-maître (avec moins de 4 bits de différence) minimisant $a_{1\to11}$ ($a_{1\to11}$=17) et ayant une valeur de $a_{2\to10}$ minimale ($a_{2\to10} = 15$). La dernière colonne indique le nombre d'octets (de bits) de clef connus aux positions des boîtes S actives au tour 1 et 11 si k^{12} est connue.

messages avec chacune des 2 clefs. De cette façon, chaque message chiffré avec K formera une paire valide avec un des messages chiffrés avec la clef liée. Comme représenté à la figure 5.10, avec 2^{8a_1} messages chiffrés sous les 2 clefs (soit 2^{8a_1+1} chiffrements) on obtient 2^{8a_1} paires qui passent les conditions du premier tour.

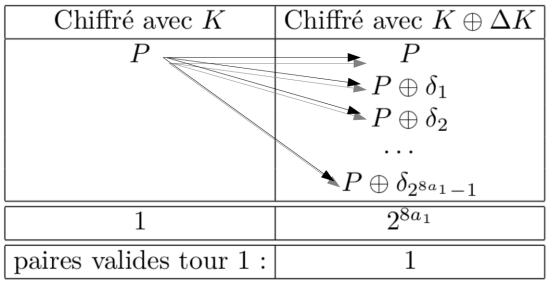


FIGURE 5.9 – En parcourant toutes les différences possibles en sortie de compression on s'assure qu'une paire est valide, au prix de $2^{8a_1} + 1$ chiffrements.

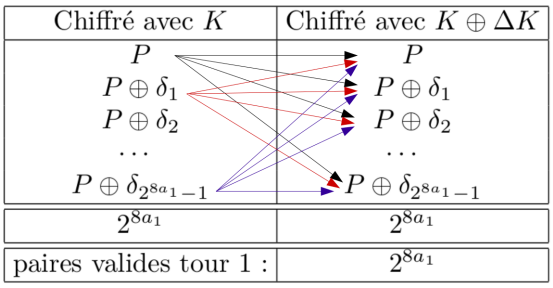


FIGURE 5.10 – Une meilleure option consiste à chiffrer ces messages avec les 2 clefs liées : on peut alors former $2^{8a_1} \times 2^{8a_1}$ paires, dont 2^{8a_1} passent les conditions du premier tour.

Au final, avec cette technique, on ne fera que $2^{7\times a_{2\to10}+1}$ chiffrements au lieu de $2^{7\times a_{1\to10}+1}$ pour obtenir une paire valide sur l'ensemble de la caractéristique.

5.4.2 Compromis possibles

Selon l'analyse que nous avons faite, le terme dominant de la complexité en temps de l'attaque est

$$C_T = 2^{7\times a_{2\to10}+1} + 2^{7\times a_{1\to11}-18.58},$$

le nombre de couples clairs-chiffrés nécessaires est :

$$C_D = 2^{7\times a_{2\to 10}+1}$$

et la mémoire requise (provenant de la technique des structures) est :

$$C_M = 2^{8\times a_1+1}$$

Ces trois complexités dépendent de deux paramètres qui sont le nombre de boîtes-S actives des tours 2 à 10, et des tours 1 à 11. La valeur minimale de ces 2 quantités est respectivement 14 et 17. Malheureusement, on ne peut pas minimiser ces deux paramètres simultanément, ce qui donne lieu à un compromis, résumé à la table 5.5 :

— Si on minimise $a_{2\to 10}$, le minimum atteignable pour $a_{1\to 11}$ est 18. Un exemple d'attaque dans une telle configuration est présenté à la figure 5.11. Cette option permet d'obtenir le meilleur compromis en termes de données nécessaires.

— Si on minimise $a_{1\to 11}$, le minimum atteignable pour $a_{2\to 10}$ est 15 et on obtient le meilleur compromis en termes de complexité en temps et en mémoire. Un exemple est donné à la figure 5.12.

$a_{2\to 10}$	$a_{1\to 11}$	C_M	C_D	C_T
14	18	2^{17}	2^{99}	$2^{107.4}$
15	17	2^{9}	2^{106}	2^{106}

TABLE 5.5 – Compromis atteignables en minimisant $a_{2\to 10}$ ou $a_{1\to 11}$.

Une des pistes que nous avons exploré pour essayer d'améliorer nos attaques consiste à réaliser une étape de recouvrement de clef recouvrant 3 tours (au lieu de 2 dans l'attaque actuelle). Ce changement permettrait d'utiliser une caractéristique différentielle sur uniquement 9 tours, donc de probabilité plus grande, et par conséquent de réduire le coût de génération des données (un des paramètres dominant de notre attaque). Cette piste n'a pas abouti car il devenait trop compliqué d'éliminer des paires ne vérifiant pas la caractéristique.

5.5 Conclusion

L'attaque décrite dans ce chapitre est la première analyse externe publiée sur PICARO, un chiffrement adapté à une utilisation dans les environnements sujets aux attaques par canaux auxiliaires. Nos travaux ont mis en avant une attaque dans le modèle à clefs liées alors que les auteurs arguaient que leur construction était résistante à ce type d'attaques. Notre analyse s'appuie tout particulièrement sur les propriétés de deux composants du chiffrement : la non-bijectivité des boîtes-S ainsi que la faible diffusion de l'algorithme de cadencement de clef. Le meilleur compromis (en terme de complexité en temps) que nous avons décrit est environ 2^{20} fois plus rapide qu'une recherche exhaustive de la clef.

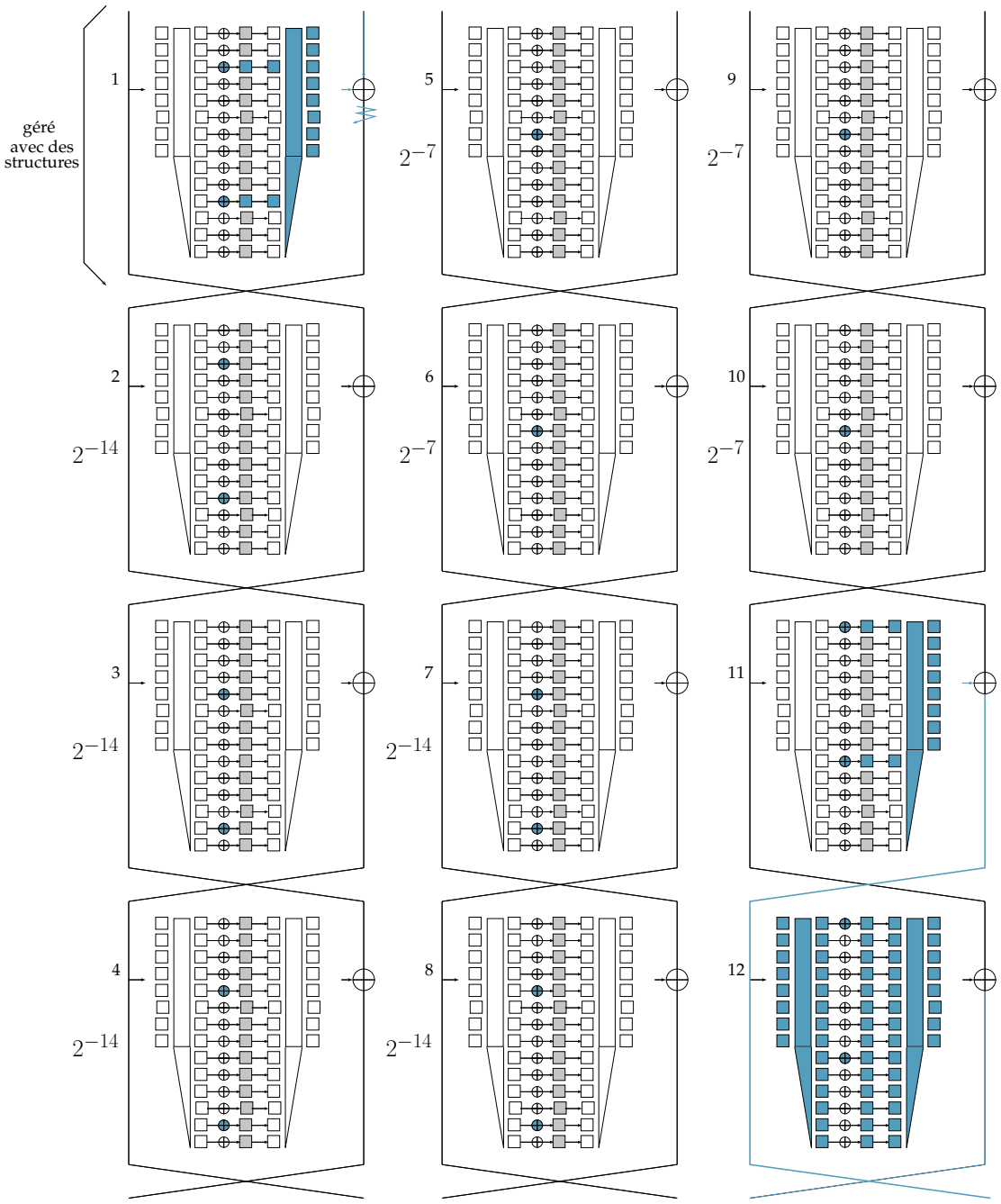


FIGURE 5.11 – Exemple d'attaque obtenue en minimisant $a_{2\to 10}$ et permettant d'obtenir le meilleur compromis en termes de complexité en données (2^{99}). La complexité en mémoire est de 2^{17} et celle en temps est de $2^{107.4}$. La différence sur la clef-maître se situe aux bits 16 et 80.

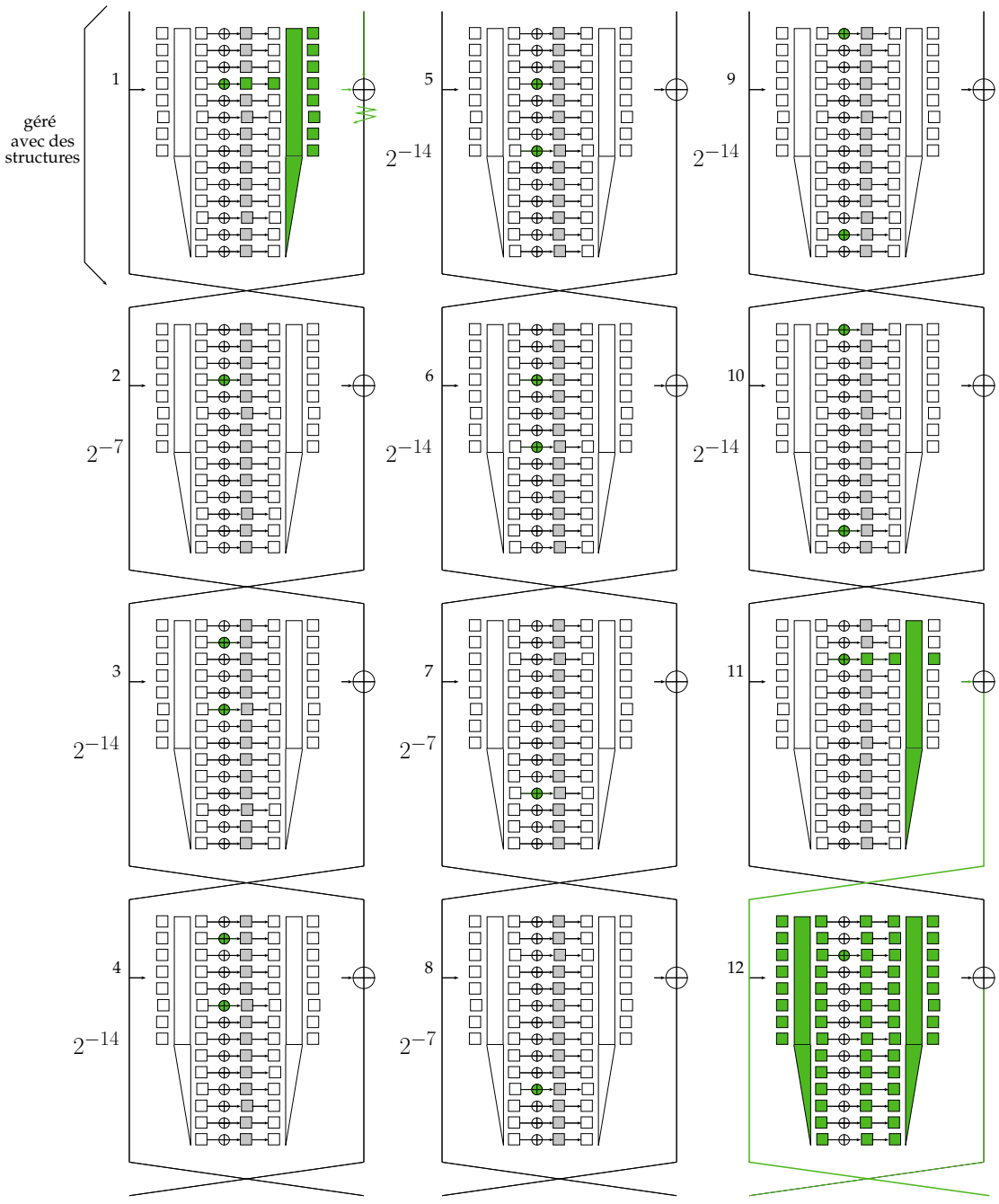


FIGURE 5.12 – Exemple d'attaque obtenue en minimisant $a_{1\to 11}$, donnant lieu au meilleur compromis en termes de complexité en temps : 2^{106} et en mémoire : 2^9 (on a $C_D = 2^{106}$). Cette configuration correspond à une différence sur la clef-maître située aux bits 27 et 123.

Deuxième partie

Cryptanalyse de chiffrements à flot

Chapitre 6

Introduction aux chiffrements à flot

La cryptographie symétrique compte deux grandes catégories de chiffrements : les chiffrements par blocs, traités dans la partie précédente, et les chiffrements à flot auxquels nous consacrons cette partie. L'objectif des chiffrements à flot est de s'approcher au plus près du *one-time-pad* ou chiffrement de Vernam : ce système consiste à réaliser un *OU exclusif* (`XOR`) bit à bit entre une séquence binaire aléatoire et le message que l'on souhaite chiffrer. Comme chaque bit est modifié avec probabilité $\frac{1}{2}$, un attaquant ne pourra pas retrouver le message clair à partir du chiffré, et cela quelle que soit sa puissance de calcul. Ce système nécessite une séquence purement aléatoire partagée par les deux parties communicantes, qui plus est aussi longue que le message que l'on souhaite chiffrer, ce qui le rend tout bonnement impraticable. La solution proposée par les chiffrements à flot est une version affaiblie de ce système et permet aux deux parties communicantes de générer la même suite pseudo-aléatoire à partir d'une clef secrète (K) et d'un vecteur d'initialisation (IV) apportés en entrée d'un même algorithme déterministe (le générateur pseudo-aléatoire).

6.1 Premières définitions

Plus formellement, un chiffrement à flot est défini par 4 fonctions :

- Une **procédure d'initialisation**, qui définit l'état initial σ^0 du générateur à partir de la clef et de l'IV,
- Une **fonction de transition**, définissant comment passer de l'état σ^t à l'instant t à celui de l'instant suivant σ^{t+1}. Cette fonction, notée f, prend le plus souvent comme unique argument l'état interne du temps précédent. Plus rarement la clef sera aussi prise en compte, et dans le cas des chiffrements auto-synchronisants (que nous définissons plus loin) cette fonction fera intervenir une partie du message chiffré,
- Une **fonction de filtrage** g, qui calcule l'élément de suite chiffrante à partir de l'état interne.
- Une fois que l'élément de suite chiffrante est produit, il est combiné avec le message (clair en chiffrement, chiffré en déchiffrement) par une **fonction de combinaison** h, qui correspondra presque toujours à un `XOR`.

Un avantage notable des chiffrements à flot en comparaison des chiffrements par blocs est qu'ils permettent le chiffrement/déchiffrement à la volée, c'est-à-dire qu'ils ne nécessitent pas d'attendre qu'un bloc complet de message soit disponible pour commencer le traitement mais

considèrent les unités [1] de message au fur et à mesure de leur réception. Cette particularité implique des protocoles plus simples pour traiter des entrées de petite taille ou de taille variable, donne lieu à des délais réduits et permet aussi de s'affranchir de la mémoire-tampon (buffer) servant à stocker le bloc de message. Comme nous le développerons dans le chapitre 7, ils sont généralement plus rapides que les chiffrements par blocs.

Le grand inconvénient de ces systèmes réside dans l'analyse de leur sécurité : alors que pour les chiffrements par blocs il existe un grand nombre d'outils pour s'assurer de leur résistance aux principales attaques, analyser la sécurité d'un chiffrement à flot est plus délicat et nécessite une analyse au cas par cas pour chaque nouvelle primitive.

On distingue deux grandes familles de chiffrements à flot :

— les **chiffrements synchrones**, pour lesquels la suite chiffrante est générée à partir de la clef et de l'IV, indépendamment du message clair et du chiffré. Pour pouvoir fonctionner, ce système nécessite que les deux parties communicantes soient parfaitement synchronisées : si un symbole est effacé ou est inséré, le déchiffrement échoue et il est nécessaire de re-synchroniser les deux générateurs en convenant d'un nouvel IV public (la clef restant inchangée). Par contre, dans le cas où un symbole est mal transmis (si sa valeur est modifiée pendant la transmission par exemple), il sera mal déchiffré mais l'avantage est que l'erreur ne se propagera pas aux symboles suivants.
Pour reprendre les notations données précédemment, un chiffrement synchrone est défini par (voir figure 6.1) :

$$\begin{aligned} \sigma^{t+1} &= f(\sigma^t, K) \\ s^t &= g(\sigma^t, K) \\ c^t &= h(s^t, m^t) \end{aligned}$$

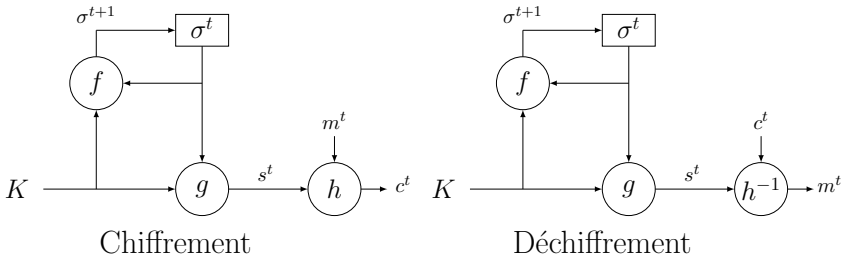


FIGURE 6.1 – Schéma générique d'un chiffrement synchrone [MvOV96]

La grande majorité des chiffrements à flot utilisés à ce jour sont de ce type et utilisent comme fonction de combinaison h le `XOR` ; c'est ce qu'on appelle des chiffrements *synchrones additifs* (figure 6.2).

— les **chiffrements auto-synchronisants** produisent la suite chiffrante à partir de la clef, de l'IV mais aussi d'un nombre b fixe de symboles du chiffré. Ils possèdent l'avantage de se re-synchroniser automatiquement : si une erreur (modification d'un symbole, effacement ou insertion) apparaît lors de la transmission du chiffré, l'algorithme de déchiffrement se trompera lors de leur traitement mais se re-synchronisera

1. Dans la plupart des cas l'unité des chiffrement à flot est le bit, mais certaines constructions travaillent sur des unités plus grandes, comme des mots de 32 bits par exemple dans le cas de SNOW [EJ00].

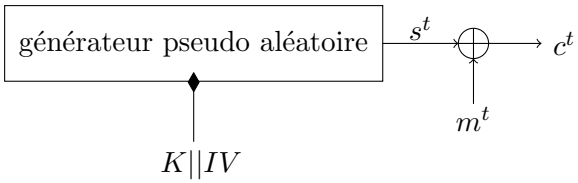


FIGURE 6.2 – Chiffrement à flot synchrone additif : un générateur pseudo-aléatoire combinant la fonction de transition f et la fonction de filtrage g est initialisé avec une clef K et un vecteur d'initialisation public (IV) et génère un bit de suite chiffrante à chaque instant. Celui-ci est additionné par `XOR` aux bits de message clair.

de lui-même après la réception de b symboles corrects. Ce type de chiffrement à flot est beaucoup plus rare que le précédent [2] et beaucoup de propositions d'instanciations ont été prouvées non sûres.
Un chiffrement auto-synchronisant a un état initial noté $\sigma^0 = (c^{-\ell}, c^{-\ell+1}, \cdots, c^{-1})$ et fonctionne avec les formules suivantes :

$$\begin{aligned} \sigma^{t+1} &= (c^{t-\ell}, c^{t+1-\ell}, \cdots, c^{t-1}) \\ s^t &= g(\sigma^t, K) \\ c^t &= h(s^t, m^t) \end{aligned}$$

Un schéma de ce type de chiffrement est donné à la figure 6.3.

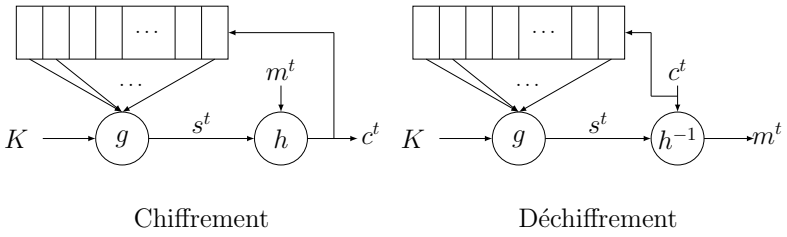


FIGURE 6.3 – Schéma générique d'un chiffrement auto-synchronisant [MvOV96]

6.2 Principes de conception des chiffrements à flot

6.2.1 Registres à décalage à rétroaction linéaire (LFSR)

Un des composants de base apparaissant dans la construction de chiffrements à flot est le registre à décalage à rétroaction linéaire ou *LFSR* [3]. Ce système, représenté à la figure 6.4, produit une suite infinie de bits [4] satisfaisant une relation de récurrence linéaire.

Le LFSR de la figure 6.4 possède L cellules contenant chacune un bit. Ces L bits forment ce que l'on appelle l'état interne du registre.

Le principe de fonctionnement du LFSR est le suivant :

— Les L cellules sont initialisées avec les valeurs $s^{L-1}, \cdots, s^0$.

2. Citons tout de même le candidat eSTREAM MOUSTIQUE.
3. Acronyme anglais pour Linear Feedback Shift Register.
4. Il est possible de généraliser les LFSR et de les placer dans un autre corps que $\mathbb{F}_2$ mais ici nous nous restreignons à ce cas qui est le plus fréquent.

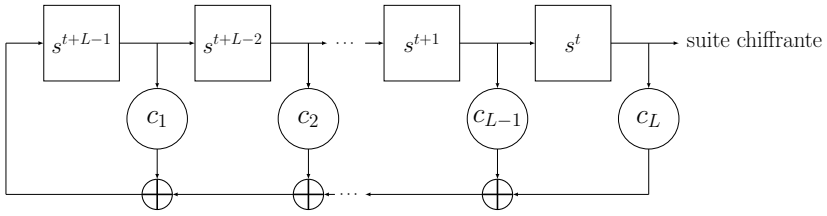


FIGURE 6.4 – Représentation générique d'un LFSR à L cellules

— À chaque instant t, le contenu d'une cellule est décalé dans la cellule directement à sa droite : le contenu de la cellule la plus à droite forme la sortie du LFSR à cet instant, tandis que la cellule la plus à gauche prend comme valeur le résultat (dans $\mathbb{F}_2$) de la formule linéaire :

$$s^{t+L} = \sum_{i=1}^{L} c_i s^{t+L-i}$$

où les c_i sont des nombres binaires appelés coefficients de rétroaction.

On associe à un LFSR un polynôme appelé polynôme de rétroaction dont les coefficients sont les c_i :

$$P(X) = 1 + \sum_{i=1}^{L} c_i X^i$$

Les propriétés de la suite engendrée par un LFSR dépend très fortement du choix des coefficients de rétroaction, et on a notamment la propriété suivante :

Propriété 6.1. *Si le polynôme de rétroaction P est primitif, alors la période de la suite engendrée (pour une initialisation non nulle) sera maximale et égale à $2^{deg(P)} - 1$. De plus, la suite produite sera équilibrée.*

Malgré cette possibilité de créer des suites de grandes périodes, les LFSR sont beaucoup trop simples — principalement du fait de leur linéarité — pour être utilisés tels quels comme générateurs pseudo-aléatoires : si le polynôme de rétroaction est connu alors l'ensemble de la suite peut être prédit à partir de L bits ; dans le cas où il est inconnu l'algorithme de Berlekamp-Massey [Mas69] permet avec une complexité en $\mathcal{O}(L^2)$ et $2L$ bits de suite chiffrante de retrouver l'initialisation et de calculer le polynôme minimal [5]. Une idée est alors d'introduire de la non-linéarité dans le générateur de sorte à le rendre plus complexe. Les deux techniques les plus répandues pour atteindre cet objectif à partir de LFSR sont les registres combinés et les registres filtrés.

6.2.2 Registres combinés

On utilise n registres en parallèle et on combine leurs sorties avec une fonction booléenne, dite fonction de combinaison (voir figure 6.5) :

$$f : \mathbb{F}_2^n \to \mathbb{F}_2.$$

5. C'est-à-dire le polynôme de rétroaction du plus petit LFSR générant la suite.

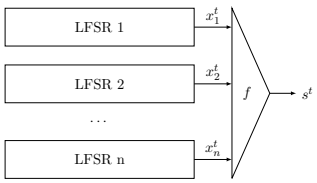


FIGURE 6.5 – Registres combinés : les sorties de plusieurs LFSR sont combinées par la fonction booléenne f, donnant la suite chiffrante.

L'exemple historique le plus connu est le générateur de Geffe [Gef73], qui utilise 3 LFSR L_1, L_2, L_3 de tailles premières entre elles et dont la suite chiffrante est construite par combinaison des sorties des 3 LFSR (notées respectivement x_1, x_2 et x_3) par la fonction $f(x_1^t, x_2^t, x_3^t) = x_1^t x_2^t + x_2^t x_3^t + x_1^t$.

6.2.3 Registres filtrés

Comme montré sur le schéma 6.6, la suite chiffrante produite par une construction par registre filtré est définie par une fonction booléenne f de certains bits de l'état interne d'un LFSR.

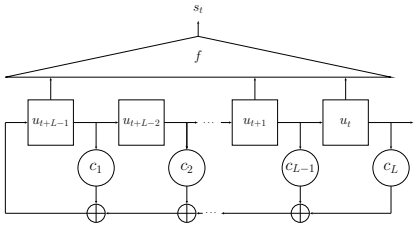


FIGURE 6.6 – Registre filtré : certains bits de l'état interne à l'instant t d'un unique LFSR servent à calculer le bit de suite chiffrante s_t en appliquant une fonction booléenne f.

Une autre possibilité consiste à considérer un registre dont la fonction de rétroaction n'est pas linéaire : c'est ce qu'on appellera un NLFSR pour *Non-Linear Feedback Shift Register*. La conception de tels registres est plus délicate.

Avant de donner un aperçu des attaques existantes sur les chiffrements à flot, donnons quelques exemples de constructions utilisées par des chiffrements à flot récents. Les chiffrements Mickey [BD08a] et Grain [HJM07] utilisent deux registres. Le premier registre est linéaire et permet d'assurer plusieurs propriétés (grande période par exemple) alors que le second registre est non-linéaire et permet de complexifier le chiffrement. Dans Mickey, les deux registres sont de plus mis à jour de façon irrégulière. Un autre exemple de proposition récente est le candidat au projet eSTREAM [6] F-FCSR [ABL+09] (pour Feedback with Carry Shift Registers), un chiffrement à flot basé sur deux registres (un principal, l'autre secondaire pour les retenues) calculant *le développement binaire du nombre p/q, où p et q sont deux entiers, q étant impair.*

6. Le projet eSTREAM (2004-2008) a été initié par le réseau d'excellence européen ECRYPT avec la vocation d'identifier de nouveaux chiffrements à flot. Il résulta dans la sélection de 7 chiffrements, 4 adaptés à des applications logicielles, 3 adaptés à des applications matérielles.

6.3 Principales attaques sur les chiffrements à flot

Différents objectifs peuvent être visés par une attaque contre un chiffrement à flot. Les trois plus courantes sont :

— les attaques par distingueur (qui consistent à réussir à reconnaître avec probabilité supérieure à $\frac{1}{2}$ si une séquence donnée est purement aléatoire ou a été générée par le chiffrement à flot),
— les attaques par recouvrement de l'état initial ou d'un état interne complet à un temps donné,
— les attaques par recouvrement de clef.

Une attaque par distingueur représente bien entendu une menace plus faible qu'une attaque retrouvant la clef puisque cette dernière permet de déchiffrer l'ensemble des messages chiffrés durant la session.

Le scénario d'attaque le plus répandu [7] correspond à l'attaque dite en clair connu : l'attaquant connaît une partie du message clair ce qui lui permet de déduire par `XOR` avec le chiffré une partie de la même taille de suite chiffrante.

Attaque par force brute/Recherche exhaustive de la clef. L'attaque par force brute est la plus simple et la plus basique : elle consiste à faire une recherche exhaustive de la clef en testant les clefs possibles les unes après les autres. Si la clef utilisée dans le chiffrement fait $|K|$ bits la complexité en temps d'une telle recherche sera donc de $2^{|K|}$ fois le coût de test [8] d'une clef.

Attaques par compromis temps-mémoire-données. Les attaques par compromis temps-mémoire-données [9] fonctionnent en deux étapes : dans la première, appelée phase de prétraitement, l'attaquant réunit des informations sur l'algorithme de chiffrement et les stocke dans une ou plusieurs tables. La seconde phase utilise ce prétraitement en le combinant à l'information (la suite chiffrante) obtenue en ligne sur le chiffrement pour retrouver la clef utilisée. Cette attaque a été introduite pour la première fois en 1980 par Martin Hellman dans son article *A cryptanalytic time-memory trade-off* [Hel80] dans le contexte des chiffrements par blocs puis fut améliorée simultanément par Golíc [Gol97] et Babbage [Bab95] dans le cas des chiffrements à flot. Elle peut être vue comme une solution intermédiaire entre l'attaque par force brute — réalisant un calcul pour chacune des clefs jusqu'à trouver la bonne, et répétant cela pour chaque instance — énormément coûteuse en temps mais ne nécessitant pas de mémoire, et l'attaque par dictionnaire — consistant à construire une énorme table contenant les résultats pour chacune des clefs — énormément coûteuse en mémoire mais qui a l'avantage de pouvoir être réutilisée plusieurs fois.

Attaques par corrélation. Les attaques par corrélation ont été introduites en 1985 par Siegenthaler [Sie85] pour cryptanalyser les générateurs par combinaison. L'idée est d'utiliser le principe dit de *diviser pour mieux régner* et de profiter du fait que l'état interne se décompose en plusieurs registres indépendants plus petits.

7. Qui est notamment celui que nous considérerons pour les cryptanalyses de Sprout et de FLIP.
8. Dans la très grande majorité des cas tester la validité d'une clef correspondra à regarder si la suite chiffrante obtenue avec la clef testée est la même que celle observée.
9. En anglais Time-Memory Data Trade-Off, parfois abrégé par TMDTO.

La première étape de l'attaque consiste à rechercher une corrélation entre la suite chiffrante et la sortie d'un des registres. Ainsi par exemple pour le générateur de Geffe introduit précédemment on peut facilement se convaincre que la valeur du bit de la suite chiffrante est égale à la valeur du bit de sortie du LFSR L_3 (même chose pour L_1) avec probabilité $\frac{3}{4}$. La seconde étape consiste alors à attaquer ce registre-ci, plus petit : on considère les unes après les autres les valeurs possibles pour son état interne, duquel on déduit la suite produite par ce registre que l'on compare avec la suite chiffrante du chiffrement entier (les s^t). Si on observe la même corrélation que celle calculée en théorie, l'état interne candidat est probablement le bon [10]. Dans le cas du générateur de Geffe, cette seconde étape correspond à réaliser une recherche exhaustive sur la valeur de l'état interne de L_3, à produire la suite des x_3^t et à conserver comme états internes possibles ceux pour lesquels on observe $x_3^t = s^t$ avec probabilité proche de $\frac{3}{4}$. On peut ensuite finaliser l'attaque en faisant une recherche exhaustive sur les registres restants ou en recommençant cette attaque avec un autre registre aussi corrélé avec la suite chiffrante.

Cette attaque se généralise à des corrélations existant entre la suite chiffrante et plusieurs registres : c'est ce qu'on appelle les corrélations d'ordre supérieur.

Attaques par corrélation rapides. Cette attaque est une amélioration de la précédente : elle consiste à réécrire le problème comme un problème de décodage [MS88].

Attaques algébriques. Les attaques algébriques [CM03] consistent à décrire le chiffrement à flot sous forme d'équations dont les inconnues sont les bits de l'état initial, puis à résoudre celui-ci. Diverses techniques peuvent être utilisées pour la résolution : si le système comporte un nombre réduit de monômes de degré supérieur ou égal à 2 on pourra utiliser une technique de linéarisation consistant à introduire une nouvelle variable pour chacun de ces monômes. Ceci permet d'obtenir un système linéaire plus simple à résoudre (voir chapitre 8). Pour d'autres situations on pourra envisager d'utiliser les bases de Gröbner [Fau99] ou des méthodes ad-hoc comme par exemple l'algorithme XL [CKPS00].

6.4 Notions de base sur les fonctions booléennes

Les attaques décrites précédemment prouvent qu'un grand soin doit être apporté au choix des fonctions utilisées dans la construction de chiffrements à flot, notamment dans la sélection des fonctions booléennes de combinaison ou de filtrage des LFSR.

Divers critères ont été introduits pour mesurer la bonne qualité d'une fonction booléenne (voir [Car10] notamment) :

— l'équilibre
— la non-linéarité (NL)
— le degré algébrique (d)
— l'immunité algébrique (AI)
— la résilience (res)

Définition 6.1. *(Fonction booléenne) : Une fonction booléenne à n variables f est une*

10. Une mauvaise hypothèse renverra un nombre de collisions proche de $\frac{1}{2}$.

fonction agissant sur des mots de n bits et à valeurs dans $\mathbb{F}_2$ *:*

$$f : \mathbb{F}_2^n \to \mathbb{F}_2$$

Il existe plusieurs façons de définir une fonction booléenne, à commencer par la *table de vérité*, contenant la liste de ses 2^n entrées possibles associées à leurs images. On peut aussi utiliser la forme algébrique normale [11].

Définition 6.2. *(ANF) Une fonction booléenne* f *est représentée de façon unique par sa forme algébrique normale, c'est-à-dire comme un polynôme à coefficients dans* $\mathbb{F}_2$ *de la forme*

$$f(x) = \sum_{u \in \mathbb{F}_2^n} f_u X^u$$

Définition 6.3. *(Équilibre) Une fonction booléenne est dite équilibrée si ses sorties sont uniformément distribuées dans* $\mathbb{F}_2$*. Cela correspond à observer autant de 0 que de 1 dans la table de vérité de* f *:*

$$\#\{x \in \mathbb{F}_2^n | f(x) = 0\} = \#\{x \in \mathbb{F}_2^n | f(x) = 1\} = 2^{n-1}$$

Cette propriété est essentielle si la fonction booléenne considérée est la fonction de filtrage d'un chiffrement à flot : si celle-ci n'est pas équilibrée il sera très simple de distinguer la suite chiffrante d'une séquence purement aléatoire qui, elle, est toujours équilibrée.

Avant d'introduire le critère de non-linéarité, rappelons la notion de distance de Hamming entre deux fonctions :

Définition 6.4. *(Distance de Hamming) La distance de Hamming entre deux fonctions booléennes en n variables,* f *et* g*, notée* $d_H(f, g)$*, permet de mesurer leur écart et correspond au nombre de fois où* f *et* g *prennent des valeurs différentes :*

$$d_H(f, g) = \#\{u \in \mathbb{F}_2^n | f(u) \neq g(u)\}$$

Définition 6.5. *(Non Linéarité) La non-linéarité d'une fonction booléenne* f *à n variables est la plus petite distance de Hamming entre* f *et toutes les fonctions affines* g *:*

$$\mathsf{NL}(f) = min_g\{d_H(f, g)\}$$

Pour améliorer la résistance aux attaques algébriques, une première idée est de s'assurer d'un haut degré algébrique.

Définition 6.6. *(Degré Algébrique) Le degré algébrique d'une fonction booléenne correspond au degré de sa forme algébrique normale.*

Cependant, ce critère n'est pas suffisant : par exemple Courtois et Meier ont montré avec leur cryptanalyse du générateur filtré Toyocrypt [CM03] qu'il était aussi possible de cryptanalyser un chiffrement utilisant une fonction booléenne de degré élevé en faisant usage d'un annulateur de f : il s'agit d'une fonction g (non nulle et idéalement de très faible degré) telle

11. Souvent abrégée en ANF, pour l'acronyme du nom anglais *Algebraic Normal Form*.

que $f(x)g(x)$ (ou $(f+1)(x)g(x)$) vaille zéro pour tout $x \in \mathbb{F}_2^n$. Si l'attaquant observe un bit de suite chiffrante égal à 1, il en déduit que l'état interne σ^t vérifie $f(\sigma^t) = 1$ donc comme $f(x)g(x) = 0 \quad \forall x \in \mathbb{F}_2^n$ on a que $g(\sigma^t) = 0$. Cette dernière égalité nous permet d'obtenir une équation en les bits de l'état interne qui est de faible degré. En en réunissant suffisamment l'attaquant peut alors former un système d'équations et monter une attaque algébrique comme précédemment, en profitant d'un plus faible degré. Cette famille d'attaques a donné lieu au critère d'immunité algébrique.

Définition 6.7. *(Immunité Algébrique) L'immunité algébrique de f, notée $\mathsf{AI}(f)$ est définie par :*

$$\mathsf{AI}(f) = \min_{g \neq 0} \{\deg(g) | fg = 0 \ \texttt{ou} \ (f \oplus 1)g = 0\}.$$

Définition 6.8. *(Résilience) Une fonction booléenne f est m-résiliente si elle reste équilibrée lorsqu'on fixe jusqu'à m de ses entrées. De façon plus générale on dira qu'une fonction booléenne est sans corrélation d'ordre m si sa distribution de valeurs ne change pas lorsque l'on fixe au plus m entrées.*

Chapitre 7

Cryptanalyse du chiffrement à flot Sprout

Ce chapitre traite de la nouvelle construction de chiffrement à flot présentée par Frederik Armknecht et Vasily Mikhalev à la conférence FSE 2015 [AM15], et plus précisément de l'instance concrète proposée dans leur article, appelée Sprout. Basée sur le chiffrement Grain, cette instance poursuit l'objectif d'être à bas coût et d'avoir une implémentation matérielle pouvant concurrencer les chiffrements du portfolio du projet eSTREAM (et plus précisément ceux du profil 2, orientés *hardware*). En revisitant un paradigme de construction issu des attaques par compromis temps/mémoire/données, les auteurs arrivent à réduire la taille de circuit de leurs constructions et annoncent une sécurité similaire à celle des constructions qui suivent le paradigme. Les résultats que nous décrivons ici, issus de recherches menées avec María Naya-Plasencia et publiés à Crypto 2015 [LN15], montrent que l'instance concrète Sprout ne fournit pas la sécurité annoncée.

7.1 Problématique ayant mené à Sprout

Comme décrit au chapitre 1, une des problématiques récentes consiste à construire un chiffrement à bas coût qui pourrait satisfaire les exigences strictes des nouvelles applications embarquées. C'est en voulant répondre à cette demande (dans le domaine hardware) et en comparant chiffrements à flot et chiffrements par blocs que Frederik Armknecht et Vasily Mikhalev ont eu l'idée décrite dans leur article. Plus précisément, leur point de départ est la remarque suivante, qui peut notamment être visualisée à la table 7.1 : lorsqu'on compare les performances en termes de taille de circuit et de débit des chiffrements à bloc et à flot, on peut voir que dans la majorité des cas le débit est bien meilleur pour les chiffrements à flot alors que la taille de circuit est plutôt mauvaise en comparaison. Leur nouvelle construction a ainsi comme objectif de surpasser ces résultats pour proposer une primitive qui aurait à la fois un débit intéressant et une faible empreinte matérielle. Parvenir à réduire la taille de circuit n'est pas trivial, car la taille des registres des chiffrements à flot provient de la règle suivante :

> Pour contrer les attaques en compromis temps/mémoire/données, la taille de l'état interne d'un chiffrement à flot doit être au minimum 2 fois la taille du paramètre de sécurité

Cipher	Taille du circuit (GE)	Débit (Kb/s)	Processus Logique (μm)	Source
PRESENT 80 [BKL+07]	1570	200	0.18	[BKL+07]
PRESENT 80 [BKL+07]	1000	11.4	0.35	[RPLP08]
KATAN-32 [CDK09]	802	12.5	0.13	[CDK09]
KATAN-48 [CDK09]	927	18.8	0.13	[CDK09]
KATAN-64 [CDK09]	1054	25.1	0.13	[CDK09]
KTANTAN-32 [CDK09]	462	12.5	0.13	[CDK09]
KTANTAN-48 [CDK09]	588	18.8	0.13	[CDK09]
KTANTAN-64 [CDK09]	688	25.1	0.13	[CDK09]
Mickey [BD08b]	3188	100	0.13	[GB08]
Trivium [Can06]	2580	100	0.13	[GB08]
Grain 80 [HJM07]	1294	100	0.13	[GB08]
Grain 80 [HJM07]	1162	100	0.18	[AM15]
Sprout [AM15]	813	100	0.18	[AM15]

TABLE 7.1 – Comparaison de la taille du circuit imprimé et du débit de chiffrement entre les finalistes eSTREAM Grain, Mickey et Trivium ainsi que Sprout, PRESENT et KATAN/KTANTAN (source : *On Lightweight Stream Ciphers with Shorter Internal States* [AM15]).

(le plus souvent correspondant à la taille de la clef).

Leur idée consiste alors à définir une nouvelle classe de chiffrements à flot pour lesquels la clef interviendrait non seulement dans la phase d'initialisation mais aussi dans la phase de génération de la suite chiffrante. Cette construction est justifiée par les auteurs par le fait que de cette façon, réaliser une attaque en TMDTO obligerait de considérer chacune des clefs possibles au moins une fois. Comme ce changement augmente la complexité des attaques par compromis, la taille de l'état interne nécessaire pour atteindre un niveau de sécurité donné sera plus petit que pour les chiffrements à flot conventionnels. De plus, le registre de clef (fixe) nécessite moins d'espace qu'un registre type LFSR, ce qui permet d'obtenir un chiffrement plus compact, donc mieux adapté à des applications à bas coût.

7.2 Spécification de l'instance concrète Sprout

7.2.1 Structure générale : l'héritage de Grain

Pour prouver la faisabilité et mesurer l'efficacité de leur nouveau schéma, Frederik Armknecht et Vasily Mikhalev ont construit une instanciation concrète de chiffrement à flot qui suit ces principes. Au lieu de partir de zéro au risque de construire un chiffrement à flot vulnérable à quelqu'une attaque, et étant donné que l'idée était de prouver la faisabilité de leur idée et non de concevoir un nouveau type de chiffrement à flot, les auteurs décidèrent de fonder leur instanciation sur un chiffrement existant, à savoir Grain [HJM07]. Ce dernier,

conçu par Martin Hell, Thomas Johansson et Willi Meier est une des familles de chiffrement à flot les plus étudiées et les plus robustes existant à ce jour : candidat à la compétition eSTREAM, Grain-v1 survécut aux différentes phases de sélection et aux nombreuses tentatives d'attaques pour finalement être retenu comme finaliste et faire partie des 3 chiffrements à flot recommandés pour un usage hardware.

Les instances de Grain[1] ont une structure générale fondée sur 2 registres de n bits (où n désigne la taille de la clef) : le premier est mis à jour de façon linéaire (LFSR) et sert à assurer la production d'une séquence binaire de période maximale, alors que le second est mis à jour de façon non-linéaire (NLFSR) et sert à rajouter de la complexité à l'ensemble du chiffrement à flot, notamment pour le protéger des attaques algébriques. Les deux registres sont agencés comme représentés à la figure 7.1 : le LFSR est mis à jour à chaque instant par une fonction linéaire f, et produit un bit qui sera combiné avec le résultat de la fonction g de mise à jour du NLFSR. À chaque étape, plusieurs bits des deux registres sont combinés de façon non-linéaire par h puis additionnés à d'autres bits pour former le bit de la suite chiffrante (ou bit de *keystream*) appelé z.

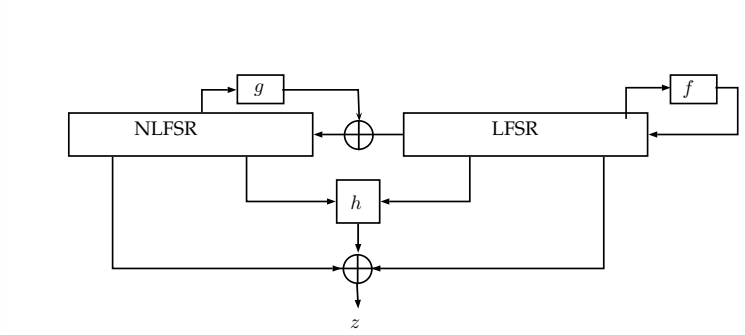


FIGURE 7.1 – Structure générale de la famille de chiffrements à flot Grain : Grain-v1 (et v0) utilisent des registres et une clef de $n = 80$ bits tandis que Grain-128 (et 128a) utilisent des registres et une clef de $n = 128$ bits.

Pour faire référence au fait que la structure de leur nouveau chiffrement se fonde sur celle de Grain, Frederik Armknecht et Vasily Mikhalev décidèrent de nommer leur instance Sprout, soit *bourgeon* en anglais.

Un schéma du fonctionnement de ce dernier est donné à la figure 7.2. On y retrouve les 2 composants principaux de Grain, c'est-à-dire un registre linéaire et un registre non-linéaire, ainsi que l'utilisation d'une fonction h pour générer le bit de la suite chiffrante z. La taille de la clef est la même que pour Grain-v0/v1 soit 80 bits.

Les deux grandes différences introduites ici en comparaison de Grain sont :

1. **La taille des registres :** Contrairement à Grain qui utilise deux registres de même taille que la clef (soit 80 bits chacun pour Grain-v0/v1), Sprout utilise deux registres de 40 bits.
2. **Le stockage et l'utilisation de la clef :** Une partie de l'état interne est dédiée au stockage de la clef, fixée pour un circuit donné. De plus, celle-ci est utilisée dans la

1. Grain compte 2 versions principales : Grain-v1, utilisant des clefs de 80 bits, successeur de Grain-v0 (qui était la soumission originale à eSTREAM), ainsi que Grain-128a, successeur de Grain-128 et nécessitant des clefs de 128 bits.

mise à jour du registre non-linéaire (c'est le bit noté k^{*t} sur la figure 7.2 et dans la suite).

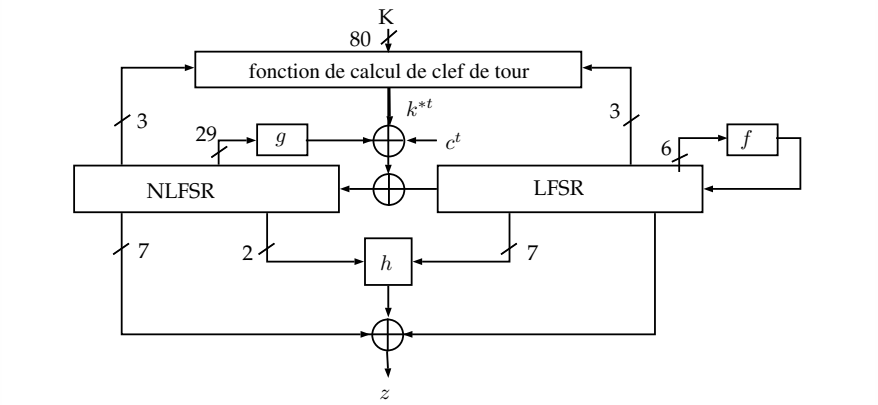


FIGURE 7.2 – Fonctionnement du chiffrement à flot Sprout.

En apparence, la taille de l'état interne de Sprout et d'une instance de Grain utilisant des clefs de 80 bits est la même : pour Sprout, on utilise 2 registres de 40 bits chacun, plus 80 bits de clef soit un total de 160 bits, et Grain utilise 2 registres de 80 bits chacun, soit là aussi 160 bits au total. Cependant, la taille effective nécessaire lors de la réalisation en hardware n'est pas la même. En effet, comme noté dans la spécification des chiffrements KATAN et KTANTAN [CDK09], stocker un état fixe (ici la clef) demande beaucoup moins d'espace qu'un registre *actif* (c'est-à-dire mis à jour périodiquement, comme c'est le cas par exemple pour un LFSR) de la même taille.

7.2.2 Définition des composants de Sprout

Dans toute la suite de cette partie, nous allons utiliser les notations et conventions suivantes, en grande partie reprises de la spécification de Sprout [AM15] :

— t numéro de tour (instant),
— $L^t = (l_0^t, l_1^t, \cdots, l_{39}^t)$ état du LFSR au temps t
— $N^t = (n_0^t, n_1^t, \cdots, n_{39}^t)$ état du NLFSR au temps t
— $iv = (iv_0, iv_1, \cdots, iv_{69})$ vecteur d'initialisation
— $k = (k_0, k_1, \cdots, k_{79})$ clef
— k^{*t} bit généré à partir de la clef, utilisé au temps t
— z^t bit de la suite chiffrante généré au temps t
— $C^t = (c_0^t, c_1^t, \cdots, c_8^t)$ état du compteur sur 9 bits au temps t

Comme schématisé à la figure 7.4, Sprout est constitué de 5 composants principaux :
— un registre linéaire (LFSR) associé à une fonction de rétroaction f
— un registre non-linéaire (NLFSR) associé à une fonction de rétroaction g
— un registre contenant la clef, associé à une fonction de calcul du bit de clef de tour
— un compteur de 9 bits
— une fonction h utilisée dans le calcul du bit de la suite chiffrante.

Chacun de ces composants est défini comme suit.

La **fonction f** intervenant dans la mise à jour du registre linéaire sert à assurer une période maximale à celui-ci. Elle est définie de façon conventionnelle pour un LFSR, c'est-à-dire à partir d'un polynôme de rétroaction primitif P :

$$P(x) = x^{40} + x^{35} + x^{25} + x^{20} + x^{15} + x^6 + 1$$

ce qui correspond à définir f de la façon suivante :

$$f(L^t) = l_0^t + l_5^t + l_{15}^t + l_{20}^t + l_{25}^t + l_{34}^t \tag{7.1}$$

On a donc $l_{39}^{t+1} = f(L^t)$.

La **fonction g** intervenant dans la mise à jour du registre non-linéaire est définie par :

$$\begin{aligned} g(N^t) \quad = \quad & n_0^t + n_{13}^t + n_{19}^t + n_{35}^t + n_{39}^t + n_2^t n_{25}^t + n_3^t n_5^t + n_7^t n_8^t + n_{14}^t n_{21}^t + n_{16}^t n_{18}^t + \\ & n_{22}^t n_{24}^t + n_{26}^t n_{32}^t + n_{33}^t n_{36}^t n_{37}^t n_{38}^t + n_{10}^t n_{11}^t n_{12}^t + n_{27}^t n_{30}^t n_{31}^t \end{aligned}$$

Fonction de calcul de la clef de tour : comme nous l'avons vu précédemment, l'originalité de Sprout réside dans l'intervention de la clef dans la mise à jour de l'état interne. Plus précisément, un bit k^{*t} est calculé à partir de la clef de la façon suivante :

$$k^{*t} = k_t \quad 0 \leq t \leq 79 \tag{7.2}$$

$$k^{*t} = (k_{t \bmod 80})(l_4^t + l_{21}^t + l_{37}^t + n_9^t + n_{20}^t + n_{29}^t) \quad t \geq 80 \tag{7.3}$$

Compteur $\mathbf{C^t}$: Outre sa fonction de compteur qui permet de déterminer le nombre de tours à exécuter dans la phase d'initialisation et de déterminer quel bit de clef doit être utilisé à chaque instant, C^t permet de fournir un bit (c_4^t) intervenant dans la mise à jour du registre non-linéaire.

Fonction de sortie : La fonction calculant la valeur du bit de suite chiffrante z^t se décompose en une partie linéaire et non linéaire. La partie non linéaire est appelée h et fait intervenir 7 bits du LFSR et 2 bits du NLFSR :

$$h(x) = n_4^t l_6^t + l_8^t l_{10}^t + l_{32}^t l_{17}^t + l_{19}^t l_{23}^t + n_4^t n_{38}^t l_{32}^t \tag{7.4}$$

La partie linéaire fait intervenir 1 bit provenant de L et 7 bits provenant de N. Le bit z^t est alors défini par :

$$z^t = h(x) + l_{30}^t + \sum_{j \in B} n_j^t \text{ avec } B = \{1, 6, 15, 17, 23, 28, 34\}. \tag{7.5}$$

7.2.3 Initialisation et génération de la suite chiffrante

Initialisation

Avant de produire la suite chiffrante, les différents registres sont initialisés avec le vecteur d'initialisation :

$$
\begin{array}{lrll}
N^0 : & n_i^0 = & iv_i & 0 \leq i \leq 39 \\
 & & & \\
L^0 : & l_i^0 = & iv_{i+40} & 0 \leq i \leq 29 \\
 & l_i^0 = & 1 & 30 \leq i \leq 38 \\
 & l_{39}^0 = & 0 &
\end{array}
$$

puis l'état est mis à jour 320 fois sans laisser sortir le bit de suite chiffrante z^t mais en utilisant celui-ci dans les calculs des bits de rétroaction des registres linéaire et non-linéaire :

$$l_{39}^{t+1} = z^t + f(L^t) \quad \text{et} \quad l_i^{t+1} = l_{i+1}^t\ , \quad 0 \leq i < 38 \tag{7.6}$$

$$n_{39}^{t+1} = z^t + k^{*t} + l_0^t + c_4^t + g(N^t) \quad \text{et} \quad n_i^{t+1} = n_{i+1}^t\ , \quad 0 \leq i < 38 \tag{7.7}$$

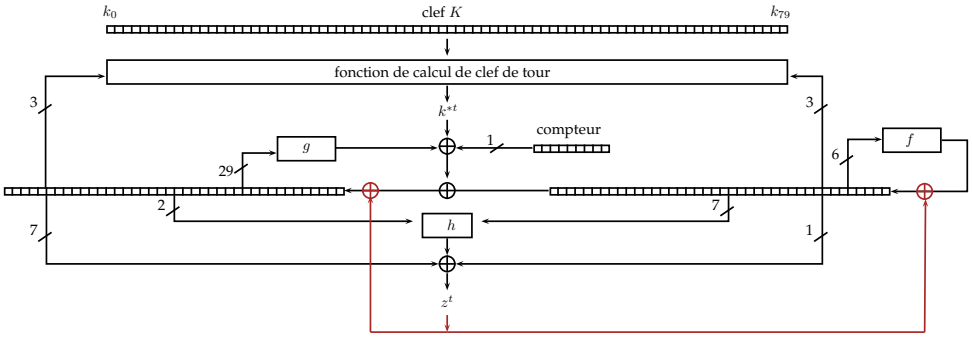


FIGURE 7.3 – Schématisation de la phase d'initialisation du chiffrement à flot Sprout.

Génération de la suite chiffrante

Lors de cette étape, la mise à jour de l'état interne suit le principe suivant (voir figure 7.4) :

$$l_{39}^{t+1} = f(L^t) \quad \text{et} \quad l_i^{t+1} = l_{i+1}^t\ , \quad 0 \leq i < 38 \tag{7.8}$$

$$n_{39}^{t+1} = k^{*t} + l_0^t + c_4^t + g(N^t) \quad \text{et} \quad n_i^{t+1} = n_{i+1}^t\ , \quad 0 \leq i < 38 \tag{7.9}$$

z^t n'est plus utilisé dans les fonctions de rétroaction mais est émis à chaque instant. Rappelons ici qu'il s'exprime par :

$$z^t = h(x) + l_{30}^t + \sum_{j \in B} n_j^t \quad \text{avec } B = \{1, 6, 15, 17, 23, 28, 34\}.$$

Notons que le nombre de bits maximum qu'un utilisateur/attaquant peut générer avec le même IV (et la même clef) est limité à 2^{40}.

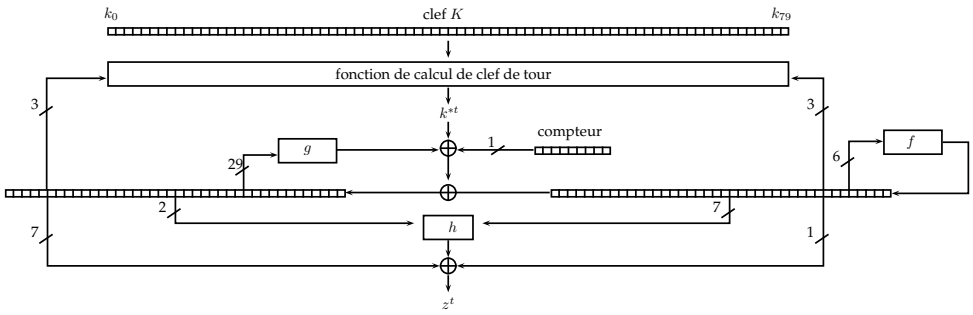


FIGURE 7.4 – Schématisation de la phase de génération de la suite chiffrante du chiffrement à flot Sprout.

7.3 Attaque

7.3.1 Premier aperçu

Scénario

Le modèle que nous considérons ici est celui de l'attaque à clair connu pour lequel une partie du message clair est à la disposition de l'attaquant et où, par conséquent, celui-ci a aussi accès à une partie de la suite chiffrante. Ce scénario d'attaque est un des plus répandus pour l'analyse des chiffrements à flot, et c'est d'ailleurs celui que nous considérerons aussi dans le chapitre suivant sur la cryptanalyse du chiffrement à flot FLIP.

Caractéristiques de Sprout exploitées dans notre attaque

Notre attaque utilise les trois caractéristiques suivantes :

1. **Faible dépendance entre le registre linéaire et le registre non-linéaire** : au vu de la description faite ci-dessus on voit que la seule influence du LFSR sur le NLFSR provient de l'intervention de l_0 dans le calcul du bit de rétroaction de N :

$$n_{39}^{t+1} = k^{*t} + l_0^t + c_4^t + g(N^t)$$

D'autre part, le LFSR devant assurer une période maximale, sa fonction de mise à jour est indépendante du NLFSR et du reste du chiffrement à flot, ce qui implique la propriété suivante, que nous allons utiliser de façon intensive dans la suite :

Propriété 7.1. *Si l'attaquant connaît la valeur des* 40 *bits de l'état interne du LFSR à un temps* t*, alors il peut calculer la valeur de n'importe quel bit du LFSR à n'importe quel instant passé ou futur.*

Démonstration. C'est une conséquence directe de la définition de la fonction de mise à jour du LFSR. □

2. **Petite taille des registres en comparaison de la taille de la clef** : N et L ne comportent tout deux que 40 bits, ce qui implique notamment qu'un attaquant peut réaliser une recherche exhaustive sur une grande fraction de ces deux registres et sur une partie de la clef sans dépasser le coût de la recherche exhaustive de cette dernière.

3. Influence non-linéaire de la clef dans la fonction de mise à jour : on a vu que la clef intervenait lors du calcul de la fonction de rétroaction de N par la formule :

$$n_{39}^{t+1} = z^t + k^{*t} + l_0^t + c_4^t + g(N^t)$$

où le bit k^{*t} est donné par (on s'intéresse uniquement à la phase de génération de la suite chiffrante) :

$$k^{*t} = (k_{t \bmod 80})(l_4^t + l_{21}^t + l_{37}^t + n_9^t + n_{20}^t + n_{29}^t)$$

Étant donné que $k_{t \bmod 80}$ intervient dans un produit, cela indique qu'à certains instants la clef n'interviendra pas. Cette remarque implique par exemple que si l'attaquant connaît la valeur des deux registres sans connaître celle de la clef, il pourra déterminer le membre $(l_4^t + l_{21}^t + l_{37}^t + n_9^t + n_{20}^t + n_{29}^t)$ et dans le cas où cette somme vaut zéro il pourra calculer la mise à jour de N.

Principe général

L'attaque que nous présentons ici utilise une technique dite de *divide-and-conquer*[2] combinée à des techniques de *guess-and-determine*[3]. Elle permet de retrouver la clef du chiffrement en utilisant seulement une centaine de bits de suite chiffrante et a une complexité en temps environ 2^{10} fois plus petite que celle de la recherche exhaustive. L'attaque se décompose en 3 étapes :

Étape 1 : la première étape consiste à construire deux listes ; la première, notée L_L, contiendra les 2^{40} valeurs possibles que peut prendre le registre L, et sera triée selon un ordre que nous détaillerons dans la suite. La seconde, appelée L_N contiendra l'ensemble des valeurs possibles pour N, couplées à d'autres bits d'information utiles à la suite de l'attaque (sa taille sera donc supérieure à 2^{40}). Là encore nous verrons dans la suite comment l'organiser de sorte à rendre l'attaque plus efficace.

Étape 2 : la seconde étape consiste à utiliser la connaissance de la suite chiffrante pour fusionner ces deux listes de sorte à obtenir un ensemble restreint de couples formés par un élément de L_L et un élément de L_N, correspondant aux valeurs possibles pour l'état interne du chiffrement. La fusion ne se fera pas de façon naïve (en envisageant chaque élément de L_L associé à chaque élément de L_N) mais en utilisant une des méthodes introduites dans [Nay11] combinée à un ensemble de cribles provenant des équations du chiffrement.

Étape 3 : une fois que l'attaquant possède un sous-ensemble d'états internes possibles, il détermine simultanément dans la dernière étape lequel correspond effectivement à l'état interne et la valeur de la clef. Cette étape nécessite comme la précédente d'utiliser l'information provenant de la suite chiffrante.

2. diviser pour mieux régner
3. supposer puis déterminer

7.3.2 Étape de fusion des listes

L'étape la plus délicate de cette attaque est la seconde : il est en effet primordial de réussir à réaliser la fusion des deux listes en conservant une complexité raisonnable et de sorte à aboutir à un ensemble restreint de possibilités pour l'état interne.

Pour pouvoir décider quelles sont les paires formées d'un élément de L_L et d'un élément de L_N qui ont une chance d'être valides, on s'intéresse à trouver un maximum de conditions sur les bits des états internes qui indiquent une (in)compatibilité. Ces conditions détermineront aussi comment arranger nos deux listes, donc nous commençons par étudier les détails de cette étape.

Pour plus de clarté et pour illustrer notre propos on utilisera un ensemble de schémas similaires au schéma 7.5, dans lequel une ligne représente un état interne du chiffrement, c'est-à-dire les 40 bits du NLFSR (à gauche) et les 40 bits du LFSR (à droite), à un instant t précisé à droite. Chaque carré représente 1 bit de l'état interne, et on numérotera en partant de la gauche.

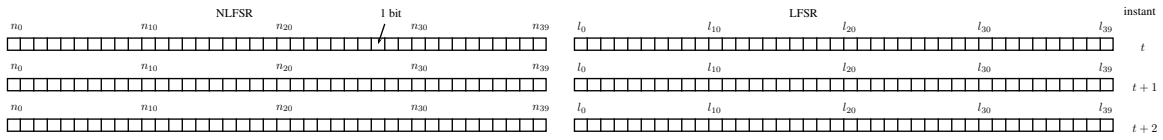


FIGURE 7.5 – Schématisation de l'état interne des deux registres aux instants t, $t+1$ et $t+2$.

On suppose ici qu'on possède une paire candidate, que l'on note (s_l^t, s_n^t) formée respectivement d'une valeur pour l'état interne du LFSR (appartenant à L_L) et d'une valeur pour l'état interne du NLFSR (appartenant à L_N) à un temps que l'on notera t. On souhaite déterminer si cette paire correspond ou non à la véritable valeur de l'état interne du chiffrement observé et pour lequel la seule information qu'on possède est la valeur de certains bits de la suite chiffrante. L'idée de base est de calculer la valeur de la suite chiffrante qu'on obtiendrait avec (s_l^t, s_n^t) (on note cette suite z^t) et de la comparer à celle observée en sortie du chiffrement que l'on souhaite attaquer (notée $\hat{z}^t$).

On rappelle que la formule donnant la valeur du bit z^t est :

$$z^t = n_4^t l_6^t + l_8^t l_{10}^t + l_{32}^t l_{17}^t + l_{19}^t l_{23}^t + n_4^t n_{38}^t l_{32}^t + l_{30}^t + n_1^t + n_6^t + n_{15}^t + n_{17}^t + n_{23}^t + n_{28}^t + n_{34}^t$$

Premier crible

Le premier test que nous pouvons réaliser pour juger de la validité de notre paire candidate est de calculer la valeur du bit z^t (c'est possible puisqu'on possède la valeur des 16 bits intervenant dans son expression) puis de le comparer avec $\hat{z}^t$. La probabilité que les deux soient différents est de 2^{-1}, donc en moyenne la moitié des candidats (s_l^t, s_n^t) seront écartés.

Extension du crible. Il est possible de réaliser deux autres tests à partir des données que nous possédons déjà. Pour visualiser cela, utilisons la représentation schématique introduite précédemment. Sur la figure 7.6 est représentée la situation de départ : l'attaquant possède une valeur candidate (s_l^t, s_n^t) correspondant à la valeur du LFSR et du NLFSR au temps t. En utilisant la définition des fonctions de mise à jour des deux registres, il déduit la valeur

de certains bits du NLFSR à des temps passés et futurs (par la formule $n_i^{t+1} = n_{i+1}^t$) et de tous les bits du LFSR à tout instant. Ces bits sont ceux colorés sur la figure 7.6.

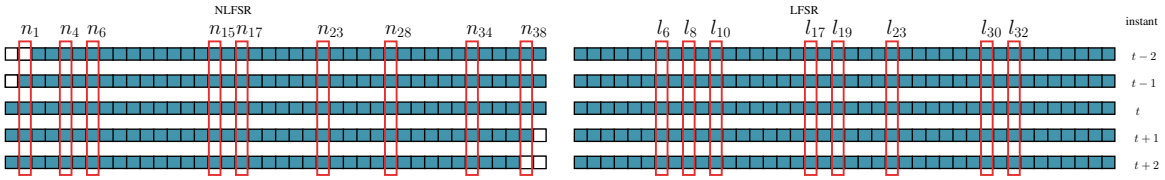


FIGURE 7.6 – Schématisation de l'état interne dans la situation de départ : tous les bits sont connus au temps t, et par extension certains bits sont connus aux temps précédents et suivants.

Les bits intervenant dans le calcul de z^t sont ceux encadrés sur la figure 7.6 : n_1, n_4, n_6, n_{15}, n_{17}, n_{23}, n_{28}, n_{34}, n_{38}, l_6, l_8, l_{10}, l_{17}, l_{19}, l_{23}, l_{30} et l_{32}. Comme nous l'avons vu précédemment, la valeur de ces bits est connue au temps t, ce qui nous permet de calculer z^t. Si on s'intéresse à la situation pour $t+1$, on peut remarquer que tous les bits nécessaires au calcul de z^{t+1} sont aussi connus : le seul bit inconnu à cet instant est n_{39}^{t+1}, et celui-ci n'intervient pas dans le calcul. Au temps $t+2$, on ne connaît ni n_{39}^{t+2} ni n_{38}^{t+2} ; or ce dernier est nécessaire au calcul de z^{t+2} : on ne peut donc pas calculer le bit de suite chiffrante qu'aurait fourni (s_l^t, s_n^t) sans la connaissance de la valeur du bit de rétroaction qui a conduit à n_{38}^{t+2}, qui dépend de la clef qui nous est inconnue.

Un raisonnement similaire pour les instants antérieurs à t montre qu'on peut calculer le bit de suite chiffrante produit à $t-1$ mais pas celui à $t-2$.

Au final, l'attaquant est capable de calculer la valeur de z à trois instants différents : t, $t+1$ et $t-1$. À chaque calcul, il compare la valeur obtenue avec celle produite par le chiffrement qu'il veut attaquer : la probabilité que ses 3 bits collisionnent avec $\hat{z}^t$, $\hat{z}^{t+1}$ et $\hat{z}^{t-1}$ et donc qu'un candidat soit considéré comme possible après ces tests est de 2^{-3}.

Définition 7.1. *(Crible 1) On appellera crible 1 l'utilisation des conditions à t, $t+1$ et $t-1$. Le nombre de bits utilisés pour ce crible sera toujours $\omega_1 = 3$.*

On voit ici que pour utiliser pleinement le crible 1 il est nécessaire de considérer un temps $t > 1$. Nous allons voir dans la suite que l'étape de recouvrement de clef fait aussi intervenir des instants antérieurs à t, ce qui impliquera au final que t correspondra à un instant environ 100 clocks après le début de génération de la suite chiffrante.

Un second crible

Comme nous venons de le voir, ce qui empêche l'attaquant de calculer plus de bits de suite chiffrante à partir de (s_l^t, s_n^t) est l'ignorance de la valeur du bit de rétroaction du NLFSR, dont le calcul dépend de la clef inconnue. Considérons par exemple le tour $t+2$. Le bit inconnu nécessaire au calcul de z^{t+2} est n_{38}^{t+2}, qui est égal à n_{39}^{t+1}, lui-même défini par :

$$n_{39}^{t+1} = k^{*t} + l_0^t + c_4^t + g(N^t)$$

dont le seul [4] terme inconnu est k^{*t}, défini par :

4. Rappelons ici que c_4^t est public et dépend uniquement du numéro de tour

$$k^{*t} = (k_{t \bmod 80})(l_4^t + l_{21}^t + l_{37}^t + n_9^t + n_{20}^t + n_{29}^t)$$

Comme évoqué à la section 7.3.1, le terme de droite du produit définissant k^{*t} est calculable et sera nul dans la moitié des cas (en moyenne) : dans ces cas-ci on pourra calculer z^{t+2}, le comparer avec $\hat{z}^{t+2}$ et ainsi obtenir un crible supplémentaire de 2^{-1}. Dans l'autre moitié des cas, le fait de ne pas connaître k^{*t} sera bloquant : il faut alors réaliser une hypothèse sur sa valeur pour pouvoir continuer. L'attaquant considère alors tour à tour les deux possibilités pour ce bit ce qui lui permet de calculer n_{39}^{t+1} et de déterminer $\hat{z}^{t+2}$. Là encore cette comparaison donnera lieu à l'élimination du couple (s_l^t, s_n^t) avec probabilité $\frac{1}{2}$, mais étant donnée l'hypothèse sur la valeur du bit de clef que le calcul a nécessité, le gain sera nul. Ces situations sont résumées à la table 7.2. L'analyse est valide pour tout instant $t + i$ où $i > 1$.

$l_4^t + l_{21}^t + l_{37}^t + n_9^t + n_{20}^t + n_{29}^t = \mathbf{0}$	$l_4^t + l_{21}^t + l_{37}^t + n_9^t + n_{20}^t + n_{29}^t = \mathbf{1}$
valeur de $k_{t \bmod 80}$ non nécessaire calcul de n_{39}^{t+1} évaluation de z^{t+2} comparaison avec $\hat{z}^{t+2}$	hypothèse sur la valeur de k^{*t} calcul de n_{39}^{t+1} évaluation de z^{t+2} comparaison avec $\hat{z}^{t+2}$
probabilité de conserver une paire : 2^{-1}	probabilité de conserver une paire : $2 \times 2^{-1} = 1$

TABLE 7.2 – Résumé des situations possibles pour le crible 2.

Récapitulons : dans la moitié des cas, la somme $l_4^t + l_{21}^t + l_{37}^t + n_9^t + n_{20}^t + n_{29}^t$ vaut zéro ce qui donne accès à un crible de facteur de réduction 2^{-1}, alors que pour l'autre moitié des cas l'effet de ce crible est annulé par le coût de l'hypothèse. La réduction obtenue par ce procédé vaut donc en moyenne :

$$F_2 = \frac{1}{2} \times 2^{-1} + \frac{1}{2} \times 1 = 2^{-0.415}.$$

Cette technique s'applique à partir de l'instant $t + 2$ et sans restriction sur son nombre d'applications. On verra dans la suite quel est le nombre optimal d'instants à considérer.

Définition 7.2. *(crible 2) On appellera crible 2 l'utilisation de ce procédé pour éliminer des paires. Le nombre de bits utilisés avec ce crible sera noté* ω_2.

Crible final. Nous allons montrer ici qu'une fois que le crible précédent a été exploité, il est possible d'écarter d'autres candidats sans réaliser d'hypothèses supplémentaires mais uniquement en utilisant ce que nous savons déjà.

Si le dernier instant traité avec le précédent crible est $\bar{t}$, la situation est celle représentée à la figure 7.7.

Au temps $\bar{t}+1$, on se retrouve dans la situation précédente où il nous manque la connaissance de $n_{38}^{\bar{t}+1}$ pour pouvoir déterminer $z^{\bar{t}+1}$. On rappelle que ce dernier est défini par :

$$\begin{aligned} z^{\bar{t}+1} \quad = \quad & n_4^{\bar{t}+1} l_6^{\bar{t}+1} + l_8^{\bar{t}+1} l_{10}^{\bar{t}+1} + l_{32}^{\bar{t}+1} l_{17}^{\bar{t}+1} + l_{19}^{\bar{t}+1} l_{23}^{\bar{t}+1} + n_4^{\bar{t}+1} n_{38}^{\bar{t}+1} l_{32}^{\bar{t}+1} + l_{30}^{\bar{t}+1} + n_1^{\bar{t}+1} \\ & + n_6^{\bar{t}+1} + n_{15}^{\bar{t}+1} + n_{17}^{\bar{t}+1} + n_{23}^{\bar{t}+1} + n_{28}^{\bar{t}+1} + n_{34}^{\bar{t}+1} \end{aligned}$$

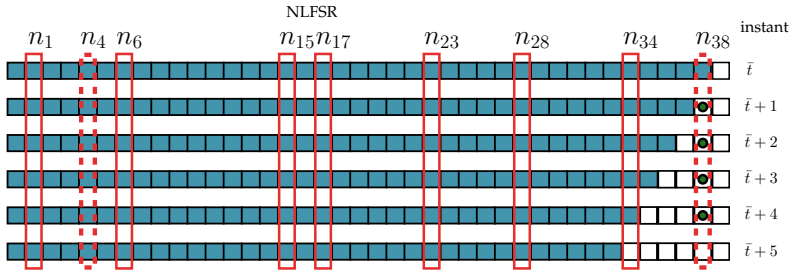


FIGURE 7.7 – Utilisation du crible 3 (le dernier instant exploité avec le crible 2 est $\bar{t}$) : sans hypothèse additionnelle, on pourra exploiter jusqu'à 4 instants supplémentaires en utilisant le fait que n_{38} intervient de façon non linéaire dans le calcul de z.

On peut voir que le bit inconnu $n_{38}^{\bar{t}+1}$ intervient une seule fois et dans un produit de 3 termes : $n_4^{\bar{t}+1} n_{38}^{\bar{t}+1} l_{32}^{\bar{t}+1}$. Les bits $n_4^{\bar{t}+1}$ et $l_{32}^{\bar{t}+1}$ sont connus de l'attaquant, et en supposant qu'ils soient uniformément distribués, la probabilité que le produit $n_4^{\bar{t}+1} \times l_{32}^{\bar{t}+1}$ soit nul vaut $1 - 2^{-2} = 2^{-0.415}$. Dans ce cas-ci, l'attaquant peut calculer la valeur de $z^{\bar{t}+1}$ sans avoir besoin de déterminer $n_{38}^{\bar{t}+1}$ et peut ainsi accéder à un crible supplémentaire de facteur de réduction 2^{-1}.

La situation est identique aux temps $\bar{t}+2$, $\bar{t}+3$ et $\bar{t}+4$, où le seul bit inconnu parmi les 17 intervenant dans le calcul de z est n_{38} : on peut donc répéter l'opération et accéder à un crible de facteur de réduction $2^{-0.415}$ pour chacun des 4 instants.

La configuration devient différente au temps $\bar{t}+5$, où en plus de $n_{38}^{\bar{t}+5}$ il manque la connaissance de $n_{34}^{\bar{t}+1}$ pour calculer $z^{\bar{t}+5}$. De plus, ce terme intervient de façon linéaire dans l'expression de z, ce qui ne nous permet pas de nous passer de sa valeur dans certains cas. On ne peut donc répéter le procédé ci-dessus que 4 fois, aux temps $\bar{t}+1$, $\bar{t}+2$, $\bar{t}+3$ et $\bar{t}+4$.

L'analyse de cette configuration est récapitulée à la table 7.3

$n_4^{t+i} l_{32}^{t+i} = 0$	$n_4^{t+i} l_{32}^{t+i} = 1$
valeur de n_{38} non nécessaire	×
calcul de z^{t+i}	×
comparaison avec $\hat{z}^{t+i}$	×
probabilité de conserver une paire : 2^{-1}	probabilité de conserver une paire : 1

TABLE 7.3 – Utilisation du crible 3 : supposons que le dernier temps utilisé avec le précédent crible est $\bar{t}$. Pour les temps $\bar{t}+i$, $i \in \{1,2,3,4\}$ et si $n_4^{t+i} l_{32}^{t+i}$ est nul on pourra calculer z et obtenir un crible supplémentaire de facteur de réduction 2^{-1}. Dans le cas contraire on conservera le candidat.

À partir de cette table on peut calculer l'efficacité moyenne du crible pour chaque instant : la probabilité de ne pas avoir besoin de n_{38} est de $\frac{3}{4}$, et dans ces cas le crible a un facteur de réduction de 2^{-1}, dans les autres cas tous les candidats sont conservés :

$$F_3 = \frac{3}{4} \times 2^{-1} + \frac{1}{4} \times 1 = 2^{-0.678}$$

Si on exploite ce crible au maximum (pour $\omega_3 = 4$ instants) le facteur de réduction total du crible sera de $2^{-2.712}$.

Définition 7.3. *(Crible 3) On appellera crible 3 l'utilisation de ce procédé pour éliminer des paires et ω_3 son nombre d'applications. On a $0 \leq \omega_3 \leq 4$.*

7.3.3 Description détaillée de l'attaque

Étape 1 : Création des listes

La première étape de l'attaque consiste à mettre en place les deux listes L_L et L_N, qui correspondent respectivement à la liste de l'ensemble des valeurs possibles pour l'état interne du registre linéaire et celui du registre non-linéaire à un temps noté t. On souhaite ensuite pouvoir décider quelles sont les paires (s_l^t, s_n^t) composées d'un élément de chaque liste qui forment un état interne possible pour le chiffrement dont on observe la suite chiffrante.

L'idée derrière la construction de ces listes est de permettre de prendre cette décision plus rapidement, notamment en offrant un accès rapide aux informations nécessaires à l'application des cribles, mais aussi en sachant de façon directe quelles parties de la table L_L nécessitent d'être analysées avec des parties de L_N.

Ainsi, comme le premier (dans l'ordre chronologique) crible que nous pouvons appliquer consiste à calculer z^{t-1}, il est intéressant de positionner en premier les valeurs des bits[5] intervenant dans son calcul, soit l_6^{t-1}, l_8^{t-1}, l_{10}^{t-1}, l_{17}^{t-1}, l_{19}^{t-1}, l_{23}^{t-1}, l_{30}^{t-1} et l_{32}^{t-1} pour L_L et n_1^{t-1}, n_4^{t-1}, n_6^{t-1}, n_{15}^{t-1}, n_{17}^{t-1}, n_{23}^{t-1}, n_{28}^{t-1}, n_{34}^{t-1} et n_{38}^{t-1} pour L_N. De la même manière, on triera ensuite les listes selon la valeur des bits intervenant dans le calcul de ω_t puis de ω_{t+1}.

On suivra le même principe pour les bits correspondant aux cribles suivants, avec la subtilité que, comme l'utilisation du crible 3 nécessite parfois une hypothèse supplémentaire sur k^*, nous allons envisager les 2 possibilités et stocker celle-ci à côté des valeurs de l'état pour tous les instants sur lequel on utilisera ce crible. Cela implique que la liste L_N sera dédoublée pour chaque hypothèse faite, et donc comportera au final $2^{40+\omega_2}$ lignes (où ω_2 correspond au nombre d'instants utilisant le crible 2).

Pour gagner de la place en mémoire et réduire la complexité des algorithmes de fusion, on calculera et on stockera le minimum d'information nécessaire à l'application des cribles. Ainsi par exemple, puisque :

$$z^t = n_4^t l_6^t + l_8^t l_{10}^t + l_{32}^t l_{17}^t + l_{19}^t l_{23}^t + n_4^t n_{38}^t l_{32}^t + l_{30}^t + n_1^t + n_6^t + n_{15}^t + n_{17}^t + n_{23}^t + n_{28}^t + n_{34}^t$$

on se restreindra à stocker uniquement 3 bits d'information dans L_N (pour chaque temps $t-1$, t, $t+1$) : n_4, n_{38} et $a = n_1 + n_6 + n_{15} + n_{17} + n_{23} + n_{28} + n_{34}$ et 3 bits dans L_L (pour chaque temps $t-1$, t, $t+1$) : l_6, l_{32} et $b = l_8 l_{10} + l_{32} l_{17} + l_{19} l_{23} + l_{30} + \hat{z}$.

On décide de faire intervenir la valeur observée du bit de suite chiffrante ($\hat{z}$) dans l'expression de b, ce qui permet de réécrire l'égalité précédente par :

$$0 = n_4^t (l_6^t + n_{38}^t l_{32}^t) + a^t + b^t$$

C'est cette égalité que l'on vérifiera lors du crible 1.

Pour les bits nécessaires à l'application du crible 2, on rajoutera la valeur de l'hypothèse sur $\kappa = k^{*t}$ dans L_N, ainsi que la valeur de la somme $\alpha = n_9 + n_{20} + n_{29}$ intervenant dans

5. Comme nous l'avons vu, ces valeurs sont facilement calculables à partir des 40 bits décrivant l'état interne.

le calcul de k^*. De la même manière on rajoutera $\beta = l_4 + l_{21} + l_{37}$ et l_0 dans L_N. Ce tri est représenté à la figure 7.8.

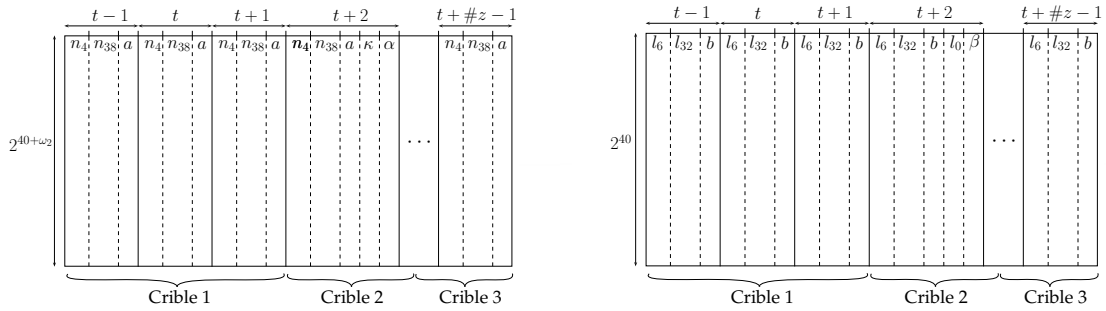


FIGURE 7.8 – Organisation des listes L_N (à gauche) et L_L (à droite).

Étape 2 : Fusion des listes L_L et L_N

Problème de fusion de listes (voir [Nay11]) : on possède deux [a] listes indépendantes L_1 et L_2 contenant des vecteurs aléatoires et uniformément distribués de s_1 et s_2 bits respectivement, ainsi qu'une fonction booléenne f agissant sur des entrées de $s_1 + s_2$ bits :

$$f : \mathbb{F}_2^{s_1+s_2} \to \mathbb{F}_2$$

On note P_f la probabilité sur l'ensemble des vecteurs $X \in \mathbb{F}_2^{s_1+s_2}$ que $f(X)$ vaille 1. Le *problème de fusion des listes conformément à une relation* correspond à établir la liste $\mathcal{L}_{\text{sol}}$ de tous les $X \in L_1 \times L_2$ vérifiant $f(X) = 1$.

a. Le problème peut bien entendu s'étendre à plus de 2 listes.

Notre situation s'assimile à ce problème : on veut fusionner les deux listes L_N et L_L pour en extraire tous les couples (s_l^t, s_n^t) vérifiant les conditions (les cribles) exprimés plus haut. Il est nécessaire d'extraire tous les couples puisque la véritable valeur de l'état interne est un des couples parmi ceux-ci : l'étape 3 de l'attaque permettra de déterminer précisément lequel.

Le problème de fusion efficace de listes a été analysé de façon détaillé dans l'article *How to Improve Rebound Attacks* [Nay11]. Cet article s'attache à améliorer la complexité en temps (et en mémoire) des attaques par rebond, en analysant une de ses étapes qui consiste à fusionner plusieurs grosses listes conformément à une relation. La complexité de cette étape dominant assez souvent celle de l'attaque, il est tout naturel de chercher à l'optimiser.

L'idée-clef de l'article [Nay11] est d'utiliser les propriétés des conditions (i.e. de la fonction f) pour fusionner plus efficacement les listes : il est montré que pour certaines formes de f, on peut énumérer les couples possibles de façon plus efficace que si l'on suit la méthode naïve consistant à réaliser le produit cartésien des deux listes puis à calculer f sur les $|L_N| \times |L_L|$ éléments obtenus.

Plusieurs algorithmes sont proposés dans [Nay11] ; celui que nous allons utiliser ici correspond au cas où la vérification de f peut se redéfinir comme la vérification simultanée de plusieurs petites fonctions et s'appelle le *Gradual Matching Algorithm* [6].

6. Ou algorithme par correspondance graduelle

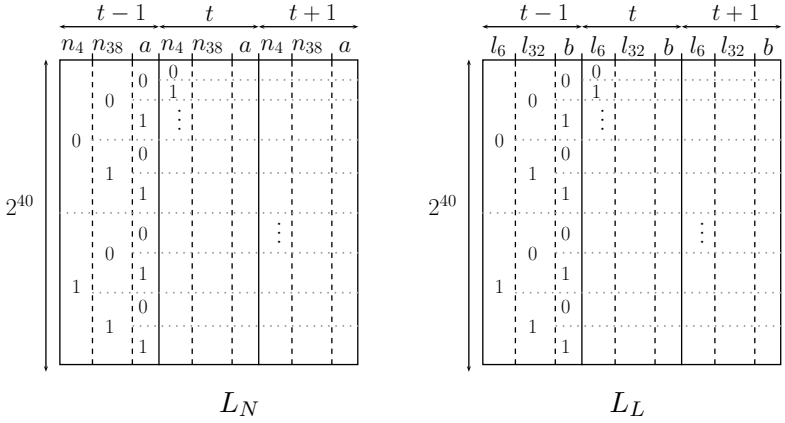
Nous allons décrire comment le mettre en œuvre lors de la description de notre attaque, en décrivant comment se déroule concrètement une attaque lorsqu'on utilise 1, 2 puis les 3 cribles possibles et comment nous trions les listes de façon optimale. Nous commençons avec le cas simple où uniquement le premier crible est appliqué :

Premier cas : Application exclusive du crible 1. Dans cette première situation, la complexité en données est de $\#z = \omega_1 + \omega_2 + \omega_3 = 3 + 0 + 0 = 3$ et on a uniquement besoin de $\hat{z}^{t-1}$, $\hat{z}^t$ et $\hat{z}^{t+1}$. Comme le crible 2 n'est pas du tout utilisé, l'attaquant n'a pas besoin de réaliser des hypothèses supplémentaires sur la valeur des bits de clefs, ce qui implique que les deux listes L_L et L_N possèdent chacune 2^{40} éléments.

Les deux listes sont classées comme suit (voir figure 7.9) :

— L_N est triée selon n_4 puis selon n_{38} et finalement selon $a = n_1 + n_6 + n_{15} + n_{17} + n_{23} + n_{28} + n_{34}$: d'abord au temps $t-1$, puis t et $t+1$.

— L_L est triée selon la valeur de l_6 puis selon celle de l_{32} et de $b = l_8 l_{10} + l_{32} l_{17} + l_{19} l_{23} + l_{30} + \hat{z}$ au temps $t-1$, t, puis $t+1$.



L_N L_L

FIGURE 7.9 – Organisation des listes L_N et L_L dans le cas où seul le crible 1 est utilisé.

On veut extraire à partir de ces deux listes l'ensemble des couples (s_l^t, s_n^t) vérifiant la propriété « *cette valeur de l'état interne peut produire* $\hat{z}^{t-1}$, $\hat{z}^t$ *et* $\hat{z}^{t+1}$ », correspondant à la vérification simultanée de 3 égalités suivantes correspondant à $\hat{z}^{t-1} = z^{t-1}$, $\hat{z}^t = z^t$ et $\hat{z}^{t+1} = z^{t+1}$:

$$n_4^{t-1}(l_6^{t-1} + n_{38}^{t-1} l_{32}^{t-1}) + a^{t-1} + b^{t-1} = 0 \quad (7.10)$$

$$n_4^{t}(l_6^{t} + n_{38}^{t} l_{32}^{t}) + a^{t} + b^{t} = 0 \quad (7.11)$$

$$n_4^{t+1}(l_6^{t+1} + n_{38}^{t+1} l_{32}^{t+1}) + a^{t+1} + b^{t+1} = 0 \quad (7.12)$$

La forme de ces équations nous indique très simplement quelles sections de L_N et de L_L étudier ensemble, et lesquelles ne valent pas la peine d'être considérées car n'importe quel couple formé d'éléments dans ces sections ne vérifiera pas les équations. Par exemple, si on s'intéresse à la première moitié de L_N correspondant à $n_4^{t-1} = 0$, on sait que, pour que l'équation 7.10 soit vérifiée, il faut que $a^{t-1} + b^{t-1}$ vaille zéro.

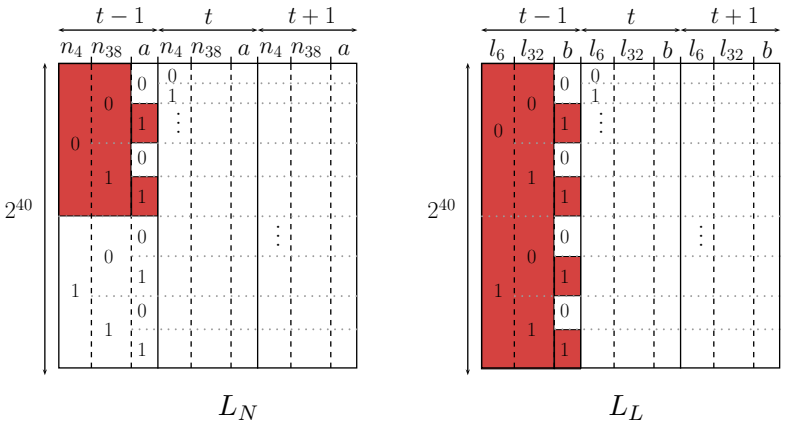


FIGURE 7.10 – Étant donné les équations, si on considère des éléments de L_N vérifiant $n_4^{t-1} = 0$ et $a^{t-1} = 1$ on sait qu'on ne peut les coupler qu'avec la moitié des éléments de L_L (ceux vérifiant $b^{t-1} = 1$).

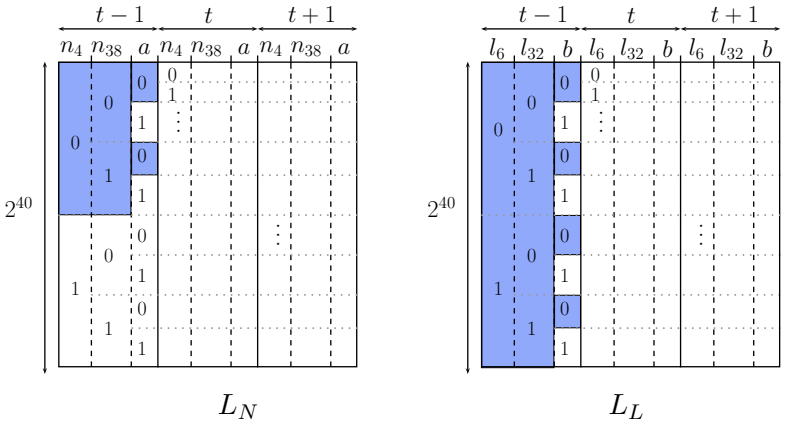


FIGURE 7.11 – Si on considère des éléments de L_N vérifiant $n_4^{t-1} = 0$ et $a^{t-1} = 0$, on ne peut les coupler qu'avec la moitié des éléments de L_L (ceux vérifiant $b^{t-1} = 0$).

Alors que l'étude naïve nous aurait fait étudier la section $n_4^{t-1} = 0$ avec l'ensemble des 2^{40} éléments de L_L, ici on ne considérera que la moitié de ces couples (voir figure 7.10 et 7.11).

Pour étudier les couples vérifiant $n_4^{t-1} = 0$, on aura une partie restreinte de L_L à étudier. On considérera ensuite les autres instants t et $t+1$ de la même manière, ce qui réduit encore plus le nombre de couples à considérer : au lieu des $2^{40} \times 2^{40} = 2^{80}$ couples on en aura uniquement 2^{77}.

Second cas : Application des cribles 1 et 2. Nous montrons ici comment se passe l'utilisation du crible 2 en plus du crible 1. Dans ce cas, on a $\omega_1 = 3$, $\omega_2 > 0$ et $\omega_3 = 0$.

La liste L_L est toujours de taille 2^{40}, mais la liste L_N est 2^{ω_2} fois plus grande que précédemment puisque, pour chaque temps postérieur à $t + 2$, l'hypothèse faite sur la valeur de k^{*t} donne lieu à un doublement du nombre d'éléments.

Nous avons vu lors de la description du crible 2 que l'hypothèse sur k^{*t} n'était pas toujours nécessaire, mais dépendait de la valeur de $l_4^t + l_{21}^t + l_{37}^t + n_9^t + n_{20}^t + n_{29}^t$: comme cette somme est liée à la valeur de certains bits de s_l (contenus dans L_L), on ne peut pas décider lors de la construction de L_N si l'hypothèse sur k^{*t} est nécessaire ou non : il faut donc la faire pour chaque élément.

À partir des listes L_L et L_N construites de la sorte, l'application du premier crible se fait de façon similaire à précédemment : grâce aux équations (7.10), (7.11) et (7.12) on sait quels couples d'éléments de L_L et L_N sont potentiellement valides, et le nombre de couples obtenus de la sorte est réduit d'un facteur de 2^{-3} par rapport à un produit cartésien naïf.

Pour appliquer le second crible, on va s'intéresser aux colonnes suivantes : la première chose à vérifier est que l'hypothèse faite sur $k^{*\tau}$ (i.e. la valeur lue dans le tableau) ne contredit pas la formule :

$$k^{*\tau} = (k_{\tau \bmod 80})(l_4^\tau + l_{21}^\tau + l_{37}^\tau + n_9^\tau + n_{20}^\tau + n_{29}^\tau) \tag{7.13}$$

Comme cela est montré dans la table 7.4, il n'y aura conformité que 3 fois sur 4, ce qui nous permet d'obtenir un crible supplémentaire de facteur de réduction $\frac{3}{4} = 2^{-0.415}$.

	$l_4^\tau + l_{21}^\tau + l_{37}^\tau + n_9^\tau + n_{20}^\tau + n_{29}^\tau = 0$	$l_4^\tau + l_{21}^\tau + l_{37}^\tau + n_9^\tau + n_{20}^\tau + n_{29}^\tau = 1$
$k^{*\tau} = 0$	valide	valide, et indique que $k_{\tau \bmod 80} = 0$
$k^{*\tau} = 1$	non valide	valide, et indique que $k_{\tau \bmod 80} = 1$

TABLE 7.4 – Probabilité qu'une hypothèse sur $k^{*\tau}$ s'avère fausse : dans 1 cas sur 4, on obtiendra une contradiction et 2 cas donneront des informations sur la clef.

Une fois qu'on a vérifié cette conformité, on calcule la valeur de z^τ que l'on compare avec $\hat{z}^\tau$: la probabilité que les deux soient égales vaut 2^{-1}, ce qui implique qu'au total, on réduira par $2^{-0.415-1} = 2^{-1.415}$ le nombre de paires formées en comparaison d'un produit naïf[7].

Troisième cas : Application de tous les cribles (1, 2 et 3). Traitons à présent le cas où le crible 3 est lui aussi pris en compte. Dans ce cas-ci, on a $\omega_1 = 3$, $\omega_2 > 0$ et $0 < \omega_3 \leq 4$ ce qui implique que l'attaquant doit avoir accès à $3 + \omega_2 + \omega_3$ bits de suite chiffrante consécutifs. Les listes L_L et L_N sont de la même taille que dans le cas où seulement le crible 1 et le crible 2 sont utilisés, soit respectivement 2^{40} et $2^{40+\omega_2}$ éléments.

Aucune hypothèse additionnelle n'est faite, mais on stockera pour chaque élément une information supplémentaire correspondant aux valeurs des bits intervenant dans le crible 3 : n_4, n_{38} et a dans L_N et l_6, l_{32} et b dans L_L aux ω_3 instants $t + \omega_2 + 2$ à $t + \omega_2 + \omega_3 + 1$ (voir figure 7.8).

La fusion graduelle réalisée avec les cribles 1 et 2 aura réduit d'un facteur $2^{-3-\omega_2 \times 0.415}$ le nombre de candidats, et la prise en compte du crible 3 ajoutera une réduction de $2^{-\omega_3 \times 0.678}$.

Étape 3 : Récupération de la clef

Les étapes 1 et 2 nous ont permis d'obtenir un ensemble réduit de $2^{80-3-\omega_2 \times 0.415-\omega_3 \times 0.678}$ états internes possibles. On souhaite à présent pouvoir décider quel est l'unique état parmi

7. Étant donné que nous avions du réaliser une hypothèse d'un bit, le facteur de réduction effectif du crible vaut donc $2^{-0.415}$.

ceux-ci qui est correct, et en déduire les 80 bits de clef-maître correspondants. On a vu lors de l'analyse de l'étape 2 que, pour chaque instant où on utilisait le crible 2, on pouvait déduire la valeur d'un bit de clef[8] dans 2 cas sur 4 pour un couple quelconque, soit dans 2 cas sur 3 si on ne considère que les cas valides. Cependant pour rendre notre analyse plus simple et comme en pratique ce nombre sera négligeable[9] nous ne prendrons pas en compte ces bits dans la suite.

L'idée centrale utilisée pour déterminer la valeur de la clef de façon simple et à un coût réduit consiste à mettre à profit le fait que la clef de tour k^{*t} intervient dans deux formules[10], qui sont :

$$k^{*t} = (k_{t \bmod 80})(l_4^t + l_{21}^t + l_{37}^t + n_9^t + n_{20}^t + n_{29}^t) \tag{7.14}$$

et

$$k^{*t} = n_{39}^{t+1} + z^t + l_0^t + c_4^t + g(N^t). \tag{7.15}$$

Ces formules montrent que si l'attaquant est en mesure de déterminer la somme $l_4^t + l_{21}^t + l_{37}^t + n_9^t + n_{20}^t + n_{29}^t$ ainsi que $n_{39}^{t+1} + z^t + l_0^t + c_4^t + g(N^t)$ il sera capable dans 1 cas sur 2 d'en déduire la valeur du bit de clef $k_{t \bmod 80}$ (voir Table 7.5). L'objectif est alors de réussir à déterminer la valeur de ces deux termes pour un maximum d'instants. Nous montrons comment y parvenir dans le paragraphe suivant.

Inverser un tour gratuitement. On suppose que les valeurs des registres linéaire et non-linéaire ont été déterminées au temps $\bar{t}$ et on admet que les bits de suite chiffrante correspondant aux états passés $\bar{t}-1$ à $\bar{t}-r$ sont connus.

Si l'attaquant connaît l'ensemble des bits de N et L à un temps $\bar{t}$, il peut déterminer par simple décalage la valeur des bits de N de l'indice i à 39 au temps $\bar{t}-i$ (voir figure 7.12).

À l'instant $\bar{t}-1$ le bit $n_0^{\bar{t}-1}$ n'est pas directement déductible à partir de $N^{\bar{t}}$, mais on sait qu'il est égal à $n_1^{\bar{t}-2}$, qui est un des bits intervenant dans l'expression de $z^{\bar{t}-2}$:

$$\begin{aligned} z^{\bar{t}-2} \;=\; & n_4^{\bar{t}-2}l_6^{\bar{t}-2} + l_8^{\bar{t}-2}l_{10}^{\bar{t}-2} + l_{32}^{\bar{t}-2}l_{17}^{\bar{t}-2} + l_{19}^{\bar{t}-2}l_{23}^{\bar{t}-2} + n_4^{\bar{t}-2}n_{38}^{\bar{t}-2}l_{32}^{\bar{t}-2} + l_{30}^{\bar{t}-2} + n_1^{\bar{t}-2} + n_6^{\bar{t}-2} \\ & + n_{15}^{\bar{t}-2} + n_{17}^{\bar{t}-2} + n_{23}^{\bar{t}-2} + n_{28}^{\bar{t}-2} + n_{34}^{\bar{t}-2} \end{aligned}$$

Comme $z^{\bar{t}-2}$ ainsi que l'ensemble des autres bits intervenant sont connus, on peut calculer la valeur de $n_1^{\bar{t}-2}$ donc obtenir la totalité du registre N au temps $\bar{t}-1$. De la même manière on utilisera d'autres bits de la suite chiffrante pour calculer d'autres positions inconnues.

Grâce à ce procédé, l'attaquant peut calculer la valeur de N à des instants $\bar{t}$ du passé, et ainsi calculer $(l_4^{\bar{t}} + l_{21}^{\bar{t}} + l_{37}^{\bar{t}} + n_9^{\bar{t}} + n_{20}^{\bar{t}} + n_{29}^{\bar{t}})$ et $n_{39}^{\bar{t}+1} + z^{\bar{t}} + l_0^{\bar{t}} + c_4^{\bar{t}} + g(N^{\bar{t}}) = k^{*\bar{t}}$.

Les différentes situations pouvant alors se produire sont résumées à la table 7.5 :

— Dans 1 cas sur 4 correspondant à $k^{*\bar{t}} = 1$ et $(l_4^{\bar{t}} + l_{21}^{\bar{t}} + l_{37}^{\bar{t}} + n_9^{\bar{t}} + n_{20}^{\bar{t}} + n_{29}^{\bar{t}}) = 0$, on obtient une contradiction puisque l'égalité (7.14) donne $k^{*t} = 0$: on peut alors éliminer cet état, il n'est pas valide et ne correspond pas à l'état ayant généré la suite chiffrante observée,

— Si la définition de k^{*t} (7.15) retourne 0 et que la somme intervenant dans (7.14) est elle aussi nulle, on ne peut pas déduire d'information sur k^t,

8. Voir table 7.4

9. Comme nous le verrons, les paramètres que nous choisissons impliquent que le nombre de bits ainsi déterminés est proche de 4, ce qui est clairement négligeable

10. Provenant de sa définition d'une part et de la formule de rétroaction du NLFSR : $n_{39}^{t+1} = z^t + k^{*t} + l_0^t + c_4^t + g(N^t)$ d'autre part

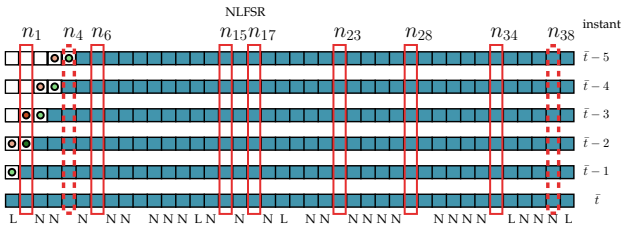


FIGURE 7.12 – On suppose connue la valeur des registres au temps $\bar{t}$. Comme montré précédemment, la valeur du registre L est connue à chaque instant : on représente donc uniquement N. Les bits encadrés en rouge sont ceux intervenant dans le calcul de z : les lignes continues désignent les bits intervenant de façon linéaire, ceux en pointillés sont ceux présents dans un produit. Si on s'intéresse au temps $\bar{t}-2$, on peut voir que tous les bits agissant dans le calcul de z sont connus, hormis n_1 : comme celui-ci intervient de façon linéaire et que $z^{\bar{t}-2}$ est donné, on peut calculer sa valeur (point vert foncé sur le schéma) et ainsi avoir accès à tout l'état au temps $\bar{t}-1$ (par simple propagation de l'information obtenue, points verts clairs sur la figure). On continue de la même manière avec $n_1^{\bar{t}-3}$ (points rouges).

— Au contraire, si k^{*t} est évalué à 0 et que $(l_4^{\bar{t}} + l_{21}^{\bar{t}} + l_{37}^{\bar{t}} + n_9^{\bar{t}} + n_{20}^{\bar{t}} + n_{29}^{\bar{t}})$ est égal à 1, on peut déduire que $k^{\bar{t}}$ vaut 0,
— Finalement, si les deux calculs donnent 1, on peut déduire que $k^{\bar{t}}$ vaut 1.

		$l_4^\tau + l_{21}^\tau + l_{37}^\tau + n_9^\tau + n_{20}^\tau + n_{29}^\tau$	
		0	1
k_τ^*	0	$\emptyset$	$k_{\tau \bmod 80} = 0$
	1	éliminer le candidat	$k_{\tau \bmod 80} = 1$

TABLE 7.5 – Lorsque l'attaquant inverse des tours, il est en mesure de calculer les deux quantités $(l_4^{\bar{t}} + l_{21}^{\bar{t}} + l_{37}^{\bar{t}} + n_9^{\bar{t}} + n_{20}^{\bar{t}} + n_{29}^{\bar{t}})$ et $n_{39}^{\bar{t}+1} + z^{\bar{t}} + l_0^{\bar{t}} + c_4^{\bar{t}} + g(N^{\bar{t}}) = k^{*\bar{t}}$: il va pouvoir en déduire de l'information sur la clef dans la moitié des cas et dans un quart des cas il va pouvoir éliminer un état candidat. Les cas restant ne lui apportent pas d'information.

Inverser r tours permet donc d'obtenir $r/2$ bits de clef et de ne conserver que $(3/4)^r$ candidats. Il est important de noter que la clef-maître est formée de 80 bits intervenant les uns après les autres par la formule

$$k^{*t} = (k_{t \bmod 80})(l_4^t + l_{21}^t + l_{37}^t + n_9^t + n_{20}^t + n_{29}^t)$$

ce qui indique qu'après 80 tours on utilisera le même bit de clef pour calculer k^{*t} : de l'information supplémentaire peut donc être obtenue si on inverse plus de 80 tours.

Traitons d'abord le cas simple pour lequel $r < 80$.

$r < 80$. Lorsqu'on inverse moins de 80 tours, chaque tour élimine un candidat avec probabilité 1/4 et donne de l'information sur un bit de clef avec probabilité 2/3, ce qui implique au final que :

— le nombre de candidats restant après r tours est de : $2^{80-3-\omega_2\times 0.415-\omega_3\times 0.678} \times (3/4)^r$
— le nombre de bits de clef déterminés pour chaque candidat est de : $r \times 2/3$

160 > r > 80. : si un état n'est pas éliminé lors de la procédure d'inversion et qu'on inverse plus de 80 tours, dans certains cas, on pourra vérifier que les informations obtenues sont concordantes. Considérons par exemple un instant τ : la probabilité que l'état candidat soit conservé et qu'on obtienne de l'information sur $k_{\tau \bmod 80}$ est de 2/3 : lorsqu'on réalise 80 tours supplémentaires, c'est le même bit de clef $k_{(\tau+80) \bmod 80} = k_{\tau \bmod 80}$ qui intervient : la probabilité d'obtenir à nouveau de l'information sur celui-ci est de 2/3, et on a en moyenne une chance sur 2 d'obtenir le même bit que précédemment.

En supposant qu'on réalise moins de 160 inversions (donc qu'un bit de clef est déterminé au plus 2 fois), on aura :

— nombre de candidats restant après les r tours : $2^{80-3-\omega_2\times 0.415-\omega_3\times 0.678} \times (3/4)^r \times (7/9)^{r-80}$

— nombre de bits de clef déterminés pour chaque candidat : $(2/3)\times(160-r)+(r-80)\times(2/3+1/3\times 2/3)$

Le calcul de la première quantité peut se voir de la façon suivante :

— Lors des 80 premiers tours, la probabilité qu'un candidat soit éliminé est de 1/4, ce qui indique qu'avant la 81ème inversion le nombre de candidats s'élèvera à

$$2^{80-3-\omega_2\times 0.415-\omega_3\times 0.678} \times (3/4)^{80}.$$

— Lorsqu'on inverse un tour supplémentaire, la probabilité de garder un candidat est encore de 3/4, mais on a une condition supplémentaire correspondant au cas où le bit de clef est déterminé une deuxième fois : un candidat sera conservé dans tous les cas hormis si la valeur retournée pour le bit est différente, c'est-à-dire avec probabilité $1 - 2/3 * 1/3 = 7/9$. Pour 81 tours le nombre de candidats restant s'élèvera donc à

$$2^{80-3-\omega_2\times 0.415-\omega_3\times 0.678} \times (3/4)^{80} \times (3/4) \times (7/9).$$

— En généralisant on retrouve bien la formule ci-dessus, soit

$$2^{80-3-\omega_2\times 0.415-\omega_3\times 0.678} \times (3/4)^r \times (7/9)^{r-80}.$$

Étant donné que chaque inversion conserve un candidat avec probabilité 3/4, au fur et à mesure que l'on inverse des tours le nombre de candidats à analyser diminue. De façon formelle le nombre d'inversions effectives sera environ de :

$$\begin{aligned} & 2^{71.8} \times \frac{1}{320} + 2^{71.8-1*0.415} \times \frac{2}{320} + \ldots + 2^{71.8-(r-1)*0.415} \times \frac{r}{320} \\ & = \frac{1}{320} \times 2^{71.8} \times \sum_{i=1}^{r} 2^{(1-i)\times 0.415} \end{aligned} \tag{7.16}$$

Pour décider quel état parmi ceux restants est correct une fois que l'on a fini d'inverser l'ensemble des r tours, on réalisera une recherche exhaustive sur les bits de clef qui n'ont pas été déterminés pendant l'attaque puis on générera la suite chiffrante pour la comparer à celle du chiffrement attaqué.

Le nombre de chiffrements réalisés sera de l'ordre de :

$$\#s \times 2^{80-\#K} \tag{7.17}$$

ce qui dans la plupart des cas sera négligeable.

Étant donné l'analyse que nous venons de donner, on peut voir que choisir $r = 100$ permet de retrouver la clef et de faire en sorte que cette étape reste moins coûteuse (du point de vue de la complexité en temps) que les précédentes. L'attaque finale nécessite donc $100+\omega_1+\omega_2+\omega_3 = 112$ bits de suite chiffrante, avec $\#s = 2^{21.41}$ et $\#K = 57.2$ ce qui amène la complexité de l'étape de recouvrement de clef à $2^{21.41} \times 2^{80-57.2} = 2^{44.2}$.

Résumé de l'attaque

Notre attaque nécessite d'inverser $r = 100$ tours pour retrouver la clef ainsi que 12 bits de suite chiffrante pour l'étape de fusion des listes permettant de trouver un nombre restreint d'états candidats. On a donc

$$C_D = 112 \text{ bits.}$$

Nos deux listes de départ, L_L et L_N, ont une taille respective de 2^{40} et $2^{40+12-4-2} = 2^{46}$ lignes. Chaque ligne consiste en un ensemble de bits correspondant à une valeur possible d'un registre (*cf.* figure 7.8). Comme ces deux listes représentent le terme dominant de notre complexité en mémoire, on a :

$$C_M < 2^{46} \text{mots.}$$

La fusion des deux listes conformément aux cribles nous permet d'obtenir $2^{71.8}$ états candidats avec $2^{69.19}$ chiffrements. L'obtention des clefs associées et l'obtention de la bonne clef sont de complexité inférieure à celle de la première étape, et l'attaque totale vérifie donc :

$$C_T \leq 2^{70} \text{chiffrements.}$$

7.4 Vérification expérimentale

Nous décrivons dans cette section l'implémentation de l'attaque que nous avons réalisée pour confirmer nos calculs de complexité. Étant donné que l'attaque que nous avons décrite a une complexité en temps non pratique, égale à 2^{70} chiffrements, nous avons été contraintes de construire une version réduite de Sprout pour mener nos expérimentations.

7.4.1 Version réduite considérée

La version réduite que nous avons construite est environ 4 fois plus petite que le chiffrement original : le LFSR et le NLFSR font tous les deux 11 bits, et la clef fait 22 bits.

Sa représentation est donnée à la figure 7.13.

Comme Sprout utilise $80 \times 4 = 320$ tours d'initialisation, nous avons fixé la durée de cette phase à $22 \times 4 = 88$ tours dans notre version.

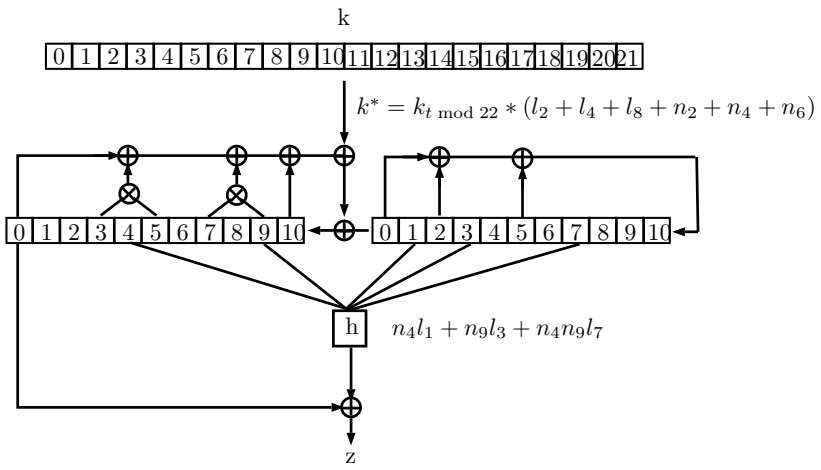


FIGURE 7.13 – Version réduite de Sprout considérée lors de notre vérification expérimentale.

Création et fusion des listes

1. Demander $r + \#z$ bits de suite chiffrante, que l'on notera z^0, z^1, …, z^{r+2}
2. Construire la première liste L_L contenant l'ensemble des valeurs possibles pour le LFSR au temps $t = r$, soit 2^{11} éléments, et la trier selon :
   — l_1^r, l_3^r et l_7^r $(t = r)$,
   — l_1^{r+1}, l_3^{r+1} et l_7^{r+1} $(t = r + 1)$,
   — l_3^{r+2} et l_7^{r+2} $(t = r + 2)$ et
   — l_0^r, $l_2^r + l_4^r + l_8^r$ $(t = r)$
3. Construire L_N contenant l'ensemble des valeurs possibles pour le NLFSR au temps $t = r$ plus la valeur d'une hypothèse, soit $2^{11+1} = 2^{12}$ lignes, et triée selon :
   — $n_0^r + z^r$, n_4^r et n_9^r $(t = r)$,
   — $n_0^{r+1} + z^{r+1}$, n_4^{r+1} et n_9^{r+1} $(t = r + 1)$,
   — $n_0^{r+2} + z^{r+2}$, n_4^{r+2} et n_9^{r+2} $(t = r + 2)$ et
   — k^{*r} $(t = r)$
4. Créer une liste M contenant la fusion des deux listes L_L et L_N selon les critères suivants :
   (a) Considérer uniquement les lignes de L_L et de L_N pour lesquels les premiers bits (l_1^r, l_3^r et l_7^r dans L_L et $n_0^r + z^r$, n_4^r et n_9^r dans L_N) vérifient l'équation définissant le bit de suite chiffrante z^r (à $t = r$) :
   $$z^r = n_4^r l_1^r + n_9^r l_3^r + n_4^r n_9^r l_7^r + n_0^r$$
      i. Pour ces états, appliquer un second crible utilisant les bits l_1^{r+1}, l_3^{r+1} et l_7^{r+1} de L_L ainsi que $n_0^{r+1} + z^{r+1}$, n_4^{r+1} et n_9^{r+1} dans L_N, pour vérifier la relation donnée par le bit z^{r+1} $(t = r + 1)$:
      $$z^{r+1} = n_4^{r+1} l_1^{r+1} + n_9^{r+1} l_3^{r+1} + n_4^{r+1} n_9^{r+1} l_7^{r+1} + n_0^{r+1}$$

A. Utiliser les troisièmes ensembles de bits pour obtenir un crible supplémentaire. On peut remarquer ici que le bit l_1^{r+2} intervenant dans ce crible est égal au bit l_3^r utilisé (donc fixé) lors de la première phase de crible. Utiliser ensuite l'information déduite du bit d'hypothèse k^{*r} et de la relation :

$$k^{*r} = k_{r \mod 22} \cdot (l_2^r + l_4^r + l_8^r + n_2^r + n_4^r + n_6^r)$$

Le facteur de réduction total obtenu avec ces cribles est de $2^{-1-1-1-0.415}$ donc le nombre de valeurs candidates restant pour les états internes du LFSR et du NLFSR à la fin de la phase de fusion est de $2^{23-3.415} = 2^{19.585}$.

Récupération de la clef

1. Pour chacun des $2^{19.585}$ états internes (LFSR et NLFSR) possibles au temps $t = r$, créer un vecteur de 22 bits noté $\tilde{K}$ et représentant la clef associée.
   (a) Pour les instants $t = r - 1$ à $t = 0$:
      i. À partir de l'état interne à $t+1$, déduire les valeurs de n_i^t, $i = 1, \cdots, 10$ et de l_i^t, $i = 1, \cdots, 10$
      ii. Calculer la valeur de n_0^t par la formule :
      $$n_0^t = z^t + n_4^t l_1^t + n_9^t l_3^t + n_4^t n_9^t l_7^t$$
      ainsi que la valeur de l_0^t donnée par la rétroaction du LFSR :
      $$l_0^t = l_2^t + l_5^t + l_{10}^{t+1}$$
      déduire de ceci la valeur de k^{*t} (donnée par la formule de rétroaction du NLFSR) :
      $$k^{*t} = n_0^t + n_3^t n_5^t + n_7^t n_9^t + n_{10}^t + l_0 + n_{10}^{t+1}$$
      iii. Calculer la valeur de $l_2^t + l_4^t + l_8^t + n_2^t + n_4^t + n_6^t$ et la combiner avec la valeur de k^{*t} obtenue dans l'étape précédente :
         A. Si $l_2^t + l_4^t + l_8^t + n_2^t + n_4^t + n_6^t = 0$ et que $k^{*t} = 1$, il y a contradiction : éliminer l'état candidat et revenir à l'étape 1
         B. Si $l_2^t + l_4^t + l_8^t + n_2^t + n_4^t + n_6^t = 1$ et que $k^{*t} = 0$, vérifier si le bit de clef a déjà été déterminé : si oui vérifier que les valeurs concordent (éliminer l'état et revenir à l'étape 1 en cas de désaccord), sinon, affecter au bit correspondant de $\tilde{K}$ la valeur 0.
         C. Si $l_2^t + l_4^t + l_8^t + n_2^t + n_4^t + n_6^t = 1$ et que $k^{*t} = 1$, vérifier si le bit de clef a déjà été déterminé : si oui vérifier que les valeurs concordent (éliminer l'état et revenir à l'étape 1 en cas de désaccord), sinon, affecter au bit correspondant de $\tilde{K}$ la valeur 1.

Nous avons implémenté et exécuté l'algorithme précédent pour différentes valeurs de r. Le nombre d'états obtenus lors de nos tests à la fin de l'étape de fusion est de $2^{19.5}$, comme prévu par la théorie. Nous avons aussi pu vérifier que l'état correct était présent dans la liste fusionnée. D'après notre analyse théorique, le nombre de candidats (formés par un état et

r	20	21	22	23	24	25	26	27	28	29	30
log du nb d'états restant à la fin de l'étape de recouvrement de clef	11.28	10.85	10.47	9.68	8.95	8.01	7.32	6.63	5.75	5.17	4.42
théorie	11.3	10.9	10.5	9.6	8.8	7.9	7.0	6.2	5.3	4.4	3.6
bits inconnus	8.68	8.02	7.30	7.12	6.96	6.77	6.67	6.32	6.29	6.03	5.94
théorie	8.7	8.0	7.3	7.1	6.9	6.7	6.4	6.2	6.0	5.8	5.6

TABLE 7.6 – Résultats expérimentaux obtenus en moyenne sur 300 états et clefs, en faisant varier r.

une clef) à la fin de l'étape de recouvrement de clef est de :

$$\begin{aligned} 2^{19.5} \times (3/4)^r &= 2^{19.5-0.415r} && \text{lorsque} && r < |k| \text{ et de} \\ 2^{19.5} \times (3/4)^r \times 2^{-(r-|k|)\times(2/3)^2} &= 2^{29.35-0.859r} && \text{lorsque} && r \geq |k|. \end{aligned}$$

et le nombre moyen de bits de clef déterminés après avoir inversé r tours sera de :

$$\begin{aligned} r \times (2/3) && \text{lorsque} && r < |k| etde \\ r \times (2/3) - (r-|k|) \times (2/3)^2 && \text{lorsque} && r \geq |k|. \end{aligned}$$

Les valeurs prises par ces formules ainsi que les valeurs obtenues lors de nos tests sont données à la table 7.6.

Une fois que les r tours ont été inversés, on rappelle qu'on détermine l'unique candidat correct en réalisant une recherche exhaustive sur les bits de clefs inconnus.

Remarque 7.1. *La complexité de l'étape de fusion des listes est de* $2^{19.585} \times \frac{3}{88} \simeq 2^{14.71}$ *chiffrements. Si on considère* $r = 26$*, le coût de l'état de recouvrement de clef sera dominé par celui de recherche exhaustive de la clef, soit un parcours de toutes les possibilités pour les* 6.67 *bits indéterminés pour chacun des* $2^{7.32}$ *états candidats restant. Le coût de test de ces* $2^{13.99}$ *états internes complets sera inférieur à celui de l'étape précédente.*

Comme on peut l'observer à la table 7.6, nos résultats expérimentaux confortent la théorie.

7.5 Conclusion

L'analyse réalisée dans ce chapitre montre que Sprout n'apporte pas la sécurité annoncée par ses concepteurs. Une des causes principales de sa faiblesse est l'intervention non-linéaire de la clef dans la mise à jour de son état interne qui, dans notre attaque, permet d'éliminer des candidats possibles pour les valeurs du LFSR et du NLFSR sans nécessiter d'hypothèse sur la clef. De nombreux autres résultats de cryptanalyse, parmi lesquels [EK15, Ban15, ZG15, MSBD15, RM16, Hao15], ont montré que Sprout n'était pas sûr. Si ces attaques indiquent que Sprout n'apporte pas une sécurité de 80 bits, elles ne révèlent cependant pas si l'idée de base des auteurs — à savoir de faire intervenir la clef dans la mise à jour de l'état interne — introduit toujours une faiblesse. La question de savoir s'il est possible de mettre en pratique

cette idée de façon sûre dans un chiffrement à flot reste donc ouverte. Pour tenter de répondre à cette question, les auteurs de Sprout associés à Christian Müller ont présenté au séminaire de cryptographie de Dagstuhl [MAM16] le chiffrement Plantlet, une variante de Sprout possédant des registres légèrement plus grands ainsi qu'une intervention de la clef plus simple. Aucune analyse externe n'a pour le moment été présentée.

Chapitre 8

Cryptanalyse de la famille de chiffrements à flot FLIP

Ce chapitre porte sur un travail réalisé avec Sébastien Duval et Yann Rotella sur le chiffrement à flot adapté aux schémas de chiffrement homomorphe appelé FLIP. Il donna lieu à l'article *Cryptanalysis of the FLIP Family of Stream Ciphers* paru à Crypto 2016 [DLR16]. Nos recherches débutèrent suite à la présentation de cette construction aux Journées Codage et Cryptographie d'octobre 2015 [Vé15] et permirent de mettre en avant la faiblesse des paramètres proposés. La communication de nos conclusions aux concepteurs leur permit de revoir leur construction pour la version finale de leur article d'Eurocrypt 2016 [MJSC16] et d'adopter un schéma résistant à notre attaque.

Ce chapitre détaille l'attaque sur la version préliminaire de FLIP que nous avons présentée à Crypto 2016 [DLR16]. Par conséquent chaque fois que nous mentionnons 'FLIP' nous faisons référence à la série de paramètres présentée aux Journées C2 et soumise à Eurocrypt 2016.

8.1 Problématique du chiffrement (totalement) homomorphe

8.1.1 Le chiffrement totalement homomorphe

Une des avancées les plus notables de ces dernières années en cryptographie asymétrique est sans conteste la description par Craig Gentry du premier système de chiffrement totalement homomorphe [Gen09].

Ce type de système a comme propriété profitable de permettre de réaliser des calculs sur des données claires en ayant uniquement accès à leurs chiffrés et en manipulant uniquement ces derniers. Il possède de nombreuses applications pratiques, à commencer par celle où un utilisateur souhaite déléguer un calcul à un serveur sans divulguer d'information sur les données en clair. Le chiffrement totalement homomorphe permet de n'envoyer au serveur que les chiffrés accompagnés de la spécification de la fonction qu'il veut voir calculer. Le serveur réalise ensuite des calculs sur les chiffrés de sorte que, lorsque l'utilisateur déchiffre le résultat, il obtienne l'évaluation souhaitée.

Le schéma proposé par Gentry repose sur l'utilisation d'un chiffrement asymétrique basé sur les réseaux euclidiens et le problème supposé difficile de recherche du plus court vecteur d'un tel réseau. Nous n'allons pas entrer dans les détails techniques ici mais il faut savoir que ce chiffrement utilise une notion d'*erreur* : un message venant d'être chiffré possède un certain

niveau d'erreur que la clef privée permet de soustraire. On peut réaliser des opérations de façon homomorphe sur les chiffrés mais chacune de ces opérations augmente le niveau d'erreur, et si celui-ci devient trop important il deviendra impossible de déchiffrer. Pour s'affranchir de cette limitation et rendre ce système totalement homomorphe donc sans restriction sur les fonctions que l'on peut appliquer aux chiffrés, Craig Gentry proposa le *bootstrapping*, une technique permettant d'atténuer le niveau d'erreur en le ramenant au niveau d'un message fraîchement chiffré. Néanmoins, cette technique demande beaucoup d'opérations et il est préférable de l'éviter autant que faire se peut.

Une première possibilité de schéma de chiffrement totalement homomorphe est donc la suivante : l'utilisatrice Alice, souhaitant déléguer un calcul à un serveur distant ou *cloud*, chiffre ses données avec le système asymétrique (et sa clef publique) et les envoie au serveur. Celui-ci réalise les opérations voulues et les renvoie à Alice qui les déchiffre avec sa clef privée.

Un des problèmes de ce système est que le chiffrement initial réalisé par Alice possède un taux d'expansion très fort [1]. Une solution proposée par la suite [NLV11] a donc été d'utiliser un chiffrement hybride qui permettrait d'éviter à Alice de supporter des opérations complexes et la transmission d'un énorme chiffré.

Le fonctionnement (schématique) de cette variante est le suivant [2] (voir figure 8.1) :

1. Alice transmet au serveur sa clef symétrique chiffrée avec le chiffrement asymétrique homomorphe : $\mathrm{HE}_{pk}(K)$.
2. Alice chiffre ses données m avec le chiffrement symétrique et envoie le résultat au serveur : $E_K(m)$.
3. Le serveur évalue homomorphiquement le déchiffrement (par le système symétrique) de $E_K(m)$. Il retrouve donc $\mathrm{HE}_{pk}(m)$ sur lequel il peut commencer à réaliser les calculs demandés par Alice.

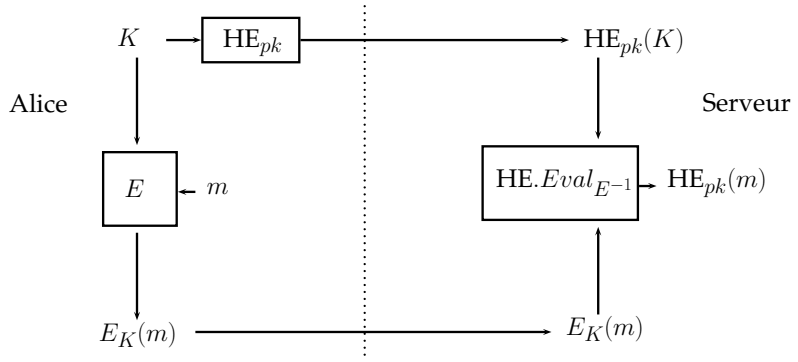


FIGURE 8.1 – Schéma de FHE hybride : Alice chiffre ses données avec le système de chiffrement symétrique et les envoie au serveur. Elle lui transmet aussi la clef symétrique, qu'elle chiffre avec le système asymétrique. De son côté, le serveur réalise l'évaluation homomorphe du déchiffrement (par le système symétrique) des données d'Alice, ce qui lui permet d'obtenir les chiffrés de ces données sous le seul chiffrement asymétrique. Il peut alors commencer à réaliser des calculs.

1. Typiquement d'environ 200 kilo-octets de chiffré pour 1 bit de clair.
2. Pour plus de détails on pourra se référer à [CCF+16] et [MJSC16].

Cette technique permet certes d'éviter le problème d'expansion du chiffré (et est donc favorable à Alice) mais implique des opérations supplémentaires du côté du serveur. Notamment, le déchiffrement que celui-ci exécute implique une croissance du bruit qu'il faut limiter au maximum.

Naturellement, on aura comme exigence pour le chiffrement symétrique qu'il possède une évaluation homomorphe de son déchiffrement qui augmente peu le bruit. Ce critère se traduira de façon différente selon les caractéristiques précises du schéma de chiffrement homomorphe considéré (on distingue notamment plusieurs générations de chiffrement homomorphe), mais correspond le plus souvent à limiter la profondeur multiplicative du chiffrement symétrique.

8.1.2 Constructions existantes

LowMC

Le premier schéma symétrique construit dans cette optique fut présenté par Martin Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen et Michael Zohner à Eurocrypt 2015 [ARS+15]. Baptisé LowMC en référence à sa faible complexité multiplicative, sa conception a été pensée pour qu'il soit particulièrement adapté à des instanciations pratiques des schémas de FHE mais aussi à deux autres types d'applications ayant une métrique similaire qui sont les schémas de *Multi-party computation* (*MPC*) et les preuves *zero-knowledge* (*ZK*) :

Définition 8.1. *(Multi-party computation) Un protocole de multi-party computation (qu'on pourrait traduire maladroitement par calcul sécurisé multi-parties) décrit comment N entités p_1, p_2, $\cdots$, p_N possédant chacune une donnée privée d_1, d_2, $\cdots$, d_N peuvent calculer une fonction publique f dépendant des N données privées, tout en gardant chacune leur donnée privée secrète.*

Pour ces 3 applications, les opérations linéaires sont peu coûteuses alors que les autres demandent un effort non négligeable : par exemple, dans le contexte du *MPC*, une opération linéaire peut être réalisée localement par chaque entité donc est très facile à réaliser contrairement à une opération non-linéaire qui impose que les parties communiquent entre elles. Comme vu précédemment, pour une application *FHE*, la problématique est très différente mais la conclusion est la même : une opération non-linéaire pose problème car elle augmente la quantité de bruit présente dans les chiffrés.

LowMC est une famille de chiffrements par blocs de type SPN : sa fonction de tour consiste en 4 opérations : l'application de boîtes-S 3×3 (nommée SboxLayer), le calcul d'une étape affine (LinearLayer) puis l'addition des constantes de tour (ConstantAddition) et finalement de la clef de tour (KeyAddition). Comme représenté figure 8.2, sa particularité réside dans le fait que l'étape non-linéaire, c'est-à-dire l'application des boîtes-S, ne concerne pas l'intégralité de l'état : une partie de l'état reste inchangé par SboxLayer. Pour assurer un haut degré de diffusion, les concepteurs ont défini l'étape LinearLayer par la multiplication de l'état avec une matrice binaire (de la taille de l'état) générée aléatoirement avec le chiffrement à flot Grain [HJM07].

L'ensemble des choix faits par les auteurs vise à minimiser 2 quantités que sont la complexité multiplicative, notée *MC* et correspondant au nombre de multiplications réalisées (ou encore au nombre de portes AND dans le cas des circuits booléens comme c'est le cas ici) et

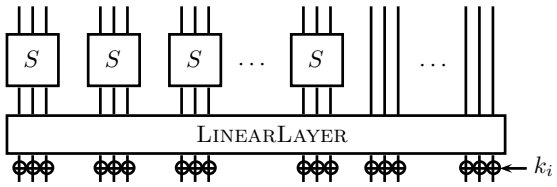


FIGURE 8.2 – Un tour du chiffrement LowMC : l'étage de boîtes-S ne recouvre que partiellement l'état pour réduire la complexité multiplicative.

la profondeur multiplicative du circuit (nombre maximal de multiplications traversées par un chemin dans le circuit), notée *ANDdepth*.

L'étude menée par les concepteurs de LowMC, prenant en compte ces métriques ainsi que la résistance du schéma aux différents types d'attaques existant les a conduit au choix des paramètres donnés dans la table 8.1.

bloc n	boîtes-S m	clef-maître k	données d	tours r	ANDdepth	ANDs pour 1 bit
256	49	80	64	12	12	6.89
256	63	128	128	14	14	10.34

TABLE 8.1 – Paramètres retenus pour 2 instanciations possibles de LowMC et permettant une sécurité de 80 ou 128 bits.

Malgré l'analyse de sécurité assez poussée réalisée par les concepteurs, plusieurs attaques sont apparues relativement peu de temps après la publication du schéma. On peut notamment évoquer la cryptanalyse d'ordre supérieure de Dobraunig, Eichlseder et Mendel [DEM15], et l'attaque par interpolation de Dinur et ses coauteurs [DLMW15].

Kreyvium et Trivium

Bien entendu, choisir un chiffrement par blocs n'est pas la seule option possible : la seconde grande famille de chiffrements symétriques que sont les chiffrements à flot a elle aussi été analysée dans le contexte particulier du chiffrement (totalement) homomorphe. Dans un article de 2015 [CCF+16], Canteaut et ses coauteurs proposent d'utiliser le finaliste eSTREAM Trivium [3] [Can06] ou une variante qu'ils spécifient dans leur article, appelée Kreyvium et qui utilise des clefs de 128 bits (au lieu de 80 bits pour Trivium).

Comme décrit dans leur article, leur solution apporte plusieurs avantages en comparaison de LowMC, à commencer par le fait que Trivium a déjà été longuement analysé sans montrer de faiblesses.

8.2 Description de la famille de chiffrements à flot FLIP

Le chiffrement auquel nous nous intéressons ici a pour objectif, comme LowMC et Trivium/Kreyvium, d'être particulièrement adapté à une intégration dans un schéma de chiffre-

3. Trivium fait partie du portfolio hardware au même titre que Grain.

ment totalement homomorphe, avec l'objectif supplémentaire d'atteindre un niveau de bruit constant **et** le plus faible possible.

8.2.1 Principe général : la structure de *filter permutator*

L'étude réalisée par les auteurs les a conduits à fonder leur système sur un chiffrement à flot de type registre filtré (voir figure 6.6) pour créer un nouveau type de construction appelé ***filter permutator*** (ce qu'on pourrait traduire par *'permutateur' filtré*). Cette construction se base sur l'utilisation d'un grand registre de taille N contenant la clef-maître K : à chaque cycle celui-ci est réordonné par une permutation pseudo-aléatoire publique puis est utilisé comme entrée de la fonction de filtrage F pour générer un bit de la suite chiffrante. Cette construction est schématisée à la figure 8.3. Elle a l'avantage de ne pas augmenter le degré algébrique de l'état interne, puisqu'elle se contente d'en permuter les positions des bits.

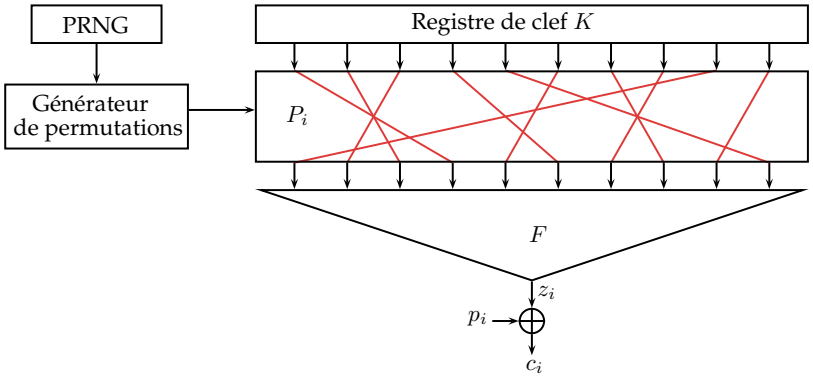


FIGURE 8.3 – Structure générale du *filter permutator*. Le *registre de clef* stocke les N bits de la clef-maître K. A chaque temps i un *générateur de permutations* paramétré par un générateur de nombres pseudo-aléatoires (ou *PRNG*) fournit une permutation P_i qui réorganise les N bits du registre. Ceux-ci sont ensuite utilisés comme entrée de la *fonction booléenne de filtrage* F servant à générer le bit de suite chiffrante z_i.

La suite chiffrante produite est ensuite utilisée de façon conventionnelle, c'est-à-dire que le i-ème bit de chiffré c_i est obtenu par l'addition modulo 2 du i-ème bit de message clair p_i avec le i-ème bit de suite chiffrante z_i :

$$c_i = p_i \oplus z_i.$$

(Le déchiffrement est bien entendu donné par $p_i = c_i \oplus z_i$.)

8.2.2 Fonction booléenne de filtrage F

Comme nous l'avons vu au début de cette partie, il existe un grand nombre d'attaques possibles sur les chiffrements à flot, et c'est pourquoi la fonction de filtrage F doit être choisie avec soin. Pour s'assurer une résistance aux attaques usuelles, les concepteurs de FLIP ont donc prêté une attention particulière aux 4 critères principaux que sont la non-linéarité (NL), la résilience (res), l'immunité algébrique (AI) et l'immunité aux attaques algébriques rapides (FAI), tout en gardant à l'esprit l'objectif d'adaptabilité aux schémas de chiffrement totalement homomorphe.

Pour atteindre ces objectifs, les auteurs décidèrent de se baser sur 3 familles de fonctions booléennes qu'ils appelèrent fonctions de *type L, Q* et *T* (respectivement pour *Linéaires*, *Quadratiques* et *Triangulaires*).

Comme précisé dans la suite, chacune de ces familles de fonctions possède un point fort vis-à-vis de certains critères mais a des caractéristiques moins bonnes pour d'autres. L'idée est alors de les combiner par somme directe, ce qui comme le montre la propriété ci-dessous permet d'obtenir une fonction avec de bonnes propriétés :

Définition 8.2. *(Somme directe) Soit* $f_1(x_0, \cdots, x_{n_1-1})$ *et* $f_2(x_{n_1}, \cdots, x_{n_1+n_2-1})$ *deux fonctions booléennes en respectivement* n_1 *et* n_2 *variables. La somme directe de* f_1 *et* f_2 *est la fonction booléenne* $f = f_1 \oplus f_2$ *en* $n_1 + n_2$ *variables définie par :*

$$f(x_0, \cdots, x_{n_1+n_2-1}) = f_1(x_0, \cdots, x_{n_1-1}) \oplus f_2(x_{n_1}, \cdots, x_{n_1+n_2-1})$$

Propriété 8.1. *La somme directe* f *de deux fonctions booléennes* f_1 *et* f_2 *possède les propriétés cryptographiques suivantes :*

- *Non-linéarité :* $\mathsf{NL}(L) = 2^{n_2}\mathsf{NL}(f_1) + 2^{n_1}\mathsf{NL}(f_2) - 2\mathsf{NL}(f_1)\mathsf{NL}(f_2)$
- *Résilience : res* (f) *= res* (f_1) *+ res* (f_2) *+ 1.*

L'immunité aux attaques algébriques (rapides) n'est pas déduite directement de celles de f_1 *et* f_2 *mais suit les inégalités suivantes :*

- *Immunité algébrique :* $\mathsf{AI}(f_1) + \mathsf{AI}(f_2) \geq \mathsf{AI}(f) \geq \max(\mathsf{AI}(f_1), \mathsf{AI}(f_2))$,
- *Immunité algébrique rapide :* $\mathsf{FAI}(f) \geq \max(\mathsf{FAI}(f_1), \mathsf{FAI}(f_2))$

On donne ici la définition et les propriétés des fonctions de type L, Q et T telles que décrites dans [MJSC16].

Définition 8.3. *(Fonction de type L) Soit* $n > 0$ *un entier positif, la n-ième fonction de type L, notée* L_n*, est la fonction booléenne en n variables définie par :*

$$L_n(x_0, \cdots, x_{n-1}) = \sum_{i=0}^{n-1} x_i.$$

Exemple : $L_3 = x_0 \oplus x_1 \oplus x_2$

Propriété 8.2. *La n-ième fonction de type L a les propriétés suivantes :*

- *Non-linéarité :* $\mathsf{NL}(L_n) = 0$,
- *Résilience :* $res(L_n) = n - 1$,
- *Immunité algébrique :* $\mathsf{AI}(L_n) = 1$,
- *Immunité algébrique rapide :* $\mathsf{FAI}(L_n) = 2$.

Définition 8.4. *(Fonction de type Q) Soit* $n > 0$ *un entier positif, la n-ième fonction de type Q, notée* Q_n*, est la fonction booléenne en* $2n$ *variables définie par :*

$$Q_n(x_0, \cdots, x_{2n-1}) = \sum_{i=0}^{n-1} x_{2i}x_{2i+1}.$$

Exemple : $Q_3 = x_0x_1 \oplus x_2x_3 \oplus x_4x_5$

Propriété 8.3. *La n-ième fonction de type Q a les propriétés suivantes :*
- *Non-linéarité :* $\mathsf{NL}(Q_n) = 2^{2n-1} - 2^{n-1}$,
- *Résilience :* $res(Q_n) = -1$, *(puisqu'elle n'est pas équilibrée)*
- *Immunité algébrique :* $\mathsf{AI}(Q_1) = 1$ *and* $\forall n > 1$, $\mathsf{AI}(Q_n) = 2$,
- *Immunité algébrique rapide :* $\mathsf{FAI}(Q_1) = 2$ *and* $\forall n > 1$, $\mathsf{FAI}(Q_n) = 4$.

Définition 8.5. *(Fonctions de type T) Soit $n > 0$ un entier positif, la n-ième fonction de type T, notée T_n, est la fonction booléenne en $\frac{n(n+1)}{2}$ variables définie par :*

$$T_n(x_0, \cdots, x_{\frac{n(n+1)}{2}-1}) = \sum_{i=1}^{n} \prod_{j=0}^{i-1} x_{j+\sum_{\ell=0}^{i-1} \ell}.$$

Exemple : $T_3 = x_0 \oplus x_1x_2 \oplus x_3x_4x_5$

Propriété 8.4. *La n-ième fonction de type T a les propriétés suivantes :*
- *Non-linéarité :*
  - $\mathsf{NL}(T_1) = 0$,
  - $\mathsf{NL}(T_{k+1}) = (2^{k+1} - 2)\mathsf{NL}(T_k) + 2^{k(k+1)/2}$.
- *Résilience :* $res(T_k) = 0$,
- *Immunité algébrique :* $\mathsf{AI}(T_k) = k$,
- *Immunité algébrique rapide :* $\mathsf{FAI}(T_k) = k + 1$.

Comme nous l'avons dit précédemment, l'idée des auteurs est de combiner par somme directe ces 3 types de fonctions booléennes pour former une fonction de filtrage F avec de bonnes propriétés cryptographiques. Nous décrivons ici la première série de paramètres concrets que les auteurs proposèrent. Les fonctions f_1, f_2 et f_3 sont définies comme suit :
- $f_1(x_0, \cdots, x_{n_1-1}) = L_{n_1}$,
- $f_2(x_{n_1}, \cdots, x_{n_1+n_2-1}) = Q_{n_2/2}$,
- $f_3(x_{n_1+n_2}, \cdots, x_{n_1+n_2+n_3-1}) = T_k$ où $n_3 = \frac{k(k+1)}{2}$.

et la fonction de filtrage F est donnée par :

$$F(x_0, \cdots, x_{n_1+n_2+n_3-1}) = L_{n_1} \oplus Q_{n_2/2} \oplus T_k.$$

8.2.3 PRNG et générateur de permutations

Le générateur de permutations choisi par les concepteurs est un Knuth shuffle [Knu69]. Cet algorithme permet de générer aléatoirement une permutation d'un ensemble fini et assure que, s'il est utilisé avec un générateur de nombres aléatoires, alors chacune des permutations de l'ensemble aura la même probabilité d'être choisie.

Données : tableau T de n éléments (indexés de 0 à $n-1$)
Résultat : T dont le contenu a été permuté
pour i *de* $n-1$ *à* 1 **faire**
  $j \leftarrow$ nombre aléatoire entre 0 et i
  échanger $T[j]$ et $T[i]$
fin

Algorithme 3 : Knuth shuffle [Knu69]

Le PRNG choisi est basé sur AES-128 (sans clef secrète) et est choisi *forward secure* (c'est-à-dire qu'un adversaire n'est pas capable de prédire les sorties précédentes même s'il arrive à obtenir l'état interne actuel du PRNG). Les permutations P_i produites par le générateur sont publiques.

8.2.4 Paramètres effectifs

Comme dans le cas des générateurs filtrés, la sécurité du *filter permutator* dépend très fortement de celle de sa fonction de filtrage, ce qui implique ici de choisir avec attention les différents paramètres N (taille de clef et nombre total de variables), n_1, n_2 et n_3 (nombre de variables de f_1, f_2 et f_3 respectivement). Les auteurs ont donc évalué la résistance de leur construction face à un grand nombre d'attaques (algébriques, algébriques rapides, par corrélation...) pour en déduire deux instances de leur schéma avec une sécurité respective de 80 et 128 bits. Ceux-ci sont donnés à la table 8.2.

Il apparaît clairement qu'étant donné la constance de l'état interne, une clef aux mauvaises propriétés sera facilement repérable et aura un impact sur l'ensemble de la suite chiffrante. Pour éviter ceci et notamment le cas des clefs possédant beaucoup de bits à 0 (ou à 1), les auteurs ont décidé de limiter l'espace des clefs uniquement à des clefs équilibrées [4], c'est-à-dire de poids $\frac{N}{2}$.

FLIP (n_1, n_2, n_3)	clef (N)	Sécurité	AI(F)	FAI(F)	res(F)	AA	FAA	HOC
FLIP (47,40,105)	192	80	14	15	47	194	88	119
FLIP (87,82,231)	400	128	21	22	87	323	136	180

TABLE 8.2 – Paramètres des deux versions de FLIP proposées ainsi que des meilleures complexités (en $\log_2$) obtenues par les concepteurs lors des analyses de sécurité vis-à-vis des attaques algébriques (AA), des attaques algébriques rapides (*fast algebraic attacks* (FAA)) et des attaques par corrélation d'ordre supérieur (*higher-order correlation attacks* (HOC)) (source : [MJSC15])

4. Section 3.5 de [MJSC16] : « *the Hamming weight of the key register is fixed (we decided to set it to $N/2$ in order to avoid weak keys* [$\cdots$]) ».

8.3 Préliminaires

8.3.1 Scénario d'attaque considéré

Scénario d'attaque. Dans l'analyse qui suit nous considérons un des scénarios les plus répandus pour l'analyse des chiffrements à flot qui est celui de l'attaque à *clair connu* pour lequel on admet qu'on possède la valeur d'une partie du message clair ($\{p_i, i \in I\}$ avec I un ensemble à déterminer [5]) ainsi que la valeur du message chiffré correspondant ($\{c_i, i \in I\}$), ce qui implique qu'on connaît la valeur d'une partie de la suite chiffrante ($\{z_i, i \in I\}$). Notre objectif est de retrouver la clef secrète utilisée dans le chiffrement, ce qui correspond ici aux N bits du registre du *filter permutator*.

Calcul des complexités. Pour exprimer les complexités de nos attaques, on utilisera la même convention que celle suivie dans la spécification de FLIP lors des évaluations de sécurité, c'est-à-dire que la complexité en temps C_T correspondra au nombre d'opérations élémentaires nécessaires pour réussir l'attaque. On exprimera la complexité en données C_D en nombre de bits de suite chiffrante nécessaires, et la complexité en mémoire, notée C_M sera exprimée en octets.

8.3.2 Vulnérabilité de FLIP face aux attaques de type *guess-and-determine*

L'attaque que nous développons ici est de type *guess-and-determine* : son principe est de faire une hypothèse sur la valeur de certains bits de l'état interne pour en déduire la valeur de bits inconnus en utilisant l'information provenant des bits de la suite chiffrante, le plus souvent par résolution d'un système d'équations.

Cette technique est très répandue et s'avère très efficace pour l'analyse de chiffrements à flot [6] mais peut aussi s'appliquer à des chiffrements par blocs comme montré dans l'article *Automatic search of attacks on round-reduced AES and applications* [BDF11].

Les deux propriétés des chiffrements FLIP qui laissent penser qu'une attaque de ce type peut s'avérer efficace sont les suivantes :

1. **La stabilité de l'état interne** : contrairement à la majorité — si ce n'est à tous — des chiffrements à flot, l'état interne n'est jamais mis à jour et reste constant pendant le chiffrement. Cela implique que si on réalise une hypothèse sur 1 bit de l'état interne à un temps t donné, celle-ci restera valable à n'importe quel temps précédent ou suivant et nous offrira constamment 1 bit d'information sur l'état interne. Si on prend l'exemple du chiffrement Sprout étudié au chapitre précédent, ceci est clairement faux : par exemple une hypothèse sur 1 bit de l'état du registre linéaire sera perdue après au plus 40 instants (en avant ou à rebours), soit à partir du moment où il intervient dans le calcul du bit de rétroaction et où d'autres bits d'information sont nécessaires.
2. **La définition de sa fonction de filtrage** : la seconde caractéristique qui semble favorable à une attaque de type guess-and-determine est la définition de la fonction de filtrage F et plus particulièrement le fait qu'elle contienne peu de monômes de haut degré. C'est ce que nous détaillons dans la section suivante.

5. Comme c'est le cas pour la majeure partie des attaques, on considérera que les $|I|$ bits sont consécutifs.
6. Elle fut notamment utilisée par Knudsen et ses coauteurs dans l'attaque de RC4 [KMP+98] ainsi que sur un grand nombre de candidats aux compétitions Nessie et eSTREAM.

8.3.3 Observations sur la fonction booléenne de filtrage F

Comme nous l'avons vu plus haut, la fonction F est formée par la somme directe de 3 fonctions booléennes f_1, f_2 et f_3, qui sont respectivement de *type* L, Q et T :

— $f_1(x_0, \cdots, x_{n_1-1}) = L_{n_1}$,
— $f_2(x_{n_1}, \cdots, x_{n_1+n_2-1}) = Q_{n_2/2}$,
— $f_3(x_{n_1+n_2}, \cdots, x_{n_1+n_2+n_3-1}) = T_k$ où $n_3 = \frac{k(k+1)}{2}$.

Cette définition implique que les monômes de degré supérieur ou égal à 3 se situent tous dans f_3, qui, rappelons-le, est définie par :

$$\begin{aligned} f_3\,(x_{n_1+n_2}, \quad \cdots \quad & , x_{n_1+n_2+n_3-1}) \\ = \quad & T_k(x_{n_1+n_2}, \cdots, x_{n_1+n_2+n_3-1}) \\ = \quad & \sum_{i=1}^{k} \prod_{j=n_1+n_2}^{n_1+n_2+i-1} x_{j+\sum_{\ell=0}^{i-1}\ell} \\ = \quad & x_{n_1+n_2} + x_{n_1+n_2+1}x_{n_1+n_2+2} + \cdots + x_{n_1+n_2+\frac{(k-1)k}{2}} \times \cdots \times x_{n_1+n_2+k-1+\frac{(k-1)k}{2}} \end{aligned}$$

où k est le degré algébrique de la fonction booléenne f_3 et vérifie $n_3 = \frac{k(k+1)}{2}$.
Autrement dit, f_3 est la somme de k monômes de degré 1 à k, chacun en des variables différentes. On a donc exactement $k-2$ monômes de degré supérieur ou égal à 3, exprimés avec $n_3 - 3$ variables différentes. Étant donné les contraintes de faible profondeur multiplicative provenant de l'application visée, k sera en pratique assez petit [7], et F possédera peu de monômes de haut degré.

Dans toute la suite on parlera de *haut* degré pour désigner un degré strictement supérieur à 2 et *bas* pour inférieur ou égal à 2.

Résumons ici les observations précédentes :
— Le registre de clef est fixe, ce qui implique qu'une hypothèse sur un bit de clef restera valide pour toute la durée de production de la suite chiffrante.
— Exactement la moitié des bits de clef sont nuls.
— Le scénario en clair connu nous assure que, pour un ensemble d'instants $t \in I$, on possède l'expression du bit de suite chiffrante z_t en fonction de la permutation publique P_t et des N bits de clef inconnus $k_0, \cdots, k_{N-1}$:

$$z_t = F(k_{P_t(0)}, k_{P_t(1)}, \cdots, k_{P_t(N-1)})), t \in I$$

— F possède de bonnes propriétés cryptographiques et un degré raisonnable (ce qui empêche toute tentative de résolution du système d'équations formé par l'ensemble des expressions des z_t) mais a très peu de monômes de haut degré.

Cet ensemble d'observations nous suggère l'idée suivante : étant donné le faible nombre de monômes de haut degré et le grand nombre de bits de clef nuls en comparaison, la probabilité que tous les monômes de haut degré soient annulés devrait être raisonnable. Dans ces cas

7. On compte 12 monômes de degré supérieur ou égal à 3 dans la version à 80 bits de sécurité (FLIP (47,40,105)) et 19 pour la version à 128 bits de sécurité (FLIP (87,82,231)).

d'annulation, l'expression de z_t sera de degré au plus 2 en les bits de clef. Si on arrive à localiser de tels instants, on sera en mesure de former un système d'équations quadratiques facile à résoudre.

Autrement dit, l'idée est de réaliser une variante de l'attaque *guess-and-determine* qui consistera à faire une hypothèse sur les indices des bits de clef nuls (et non sur la valeur d'un ensemble de bits de clef fixés) pour en déduire un ensemble d'instants t pour lesquels z_t fournit une équation quadratique en les bits de clefs restants. Le faible degré du système rend sa résolution facile (par exemple avec des techniques de linéarisation), et celle-ci permet de retrouver l'ensemble de la clef.

8.3.4 Probabilité d'annuler l'ensemble des monômes de haut degré de F

Pour déterminer si l'idée précédente a une chance de mener à une attaque, nous devons commencer par nous assurer que la probabilité que l'expression de z_t soit quadratique, étant donné un nombre fixé ℓ de bits de clef nuls, a une probabilité suffisamment élevée.

Cette quantité est primordiale puisqu'elle détermine la taille de la suite chiffrante à laquelle doit avoir accès l'attaquant pour obtenir suffisamment d'équations quadratiques pour pouvoir résoudre le système. Bien que dans la spécification initiale du chiffrement [MJSC15] aucune limite ne soit donnée sur le nombre de bits qu'un utilisateur ou qu'un attaquant peut produire, une limite communément admise est de générer moins de bits que la racine de la taille de l'état interne, soit ici $2^{\frac{192}{2}} = 2^{96}$ bits. Il est donc important de s'assurer que les paramètres de l'attaque nous permettent de rester en deçà [8].

Plus formellement on cherche à déterminer la probabilité qu'une permutation aléatoire P_i positionne les ℓ bits supposés nuls de la clef de sorte que l'expression de z_i en les bits de clef restants (inconnus) soit un polynôme de degré au plus 2. On note cette probabilité $\mathbb{P}_\ell$.

Premier cas : $\ell < k - 2$. Comme nous l'avons vu précédemment, il y a exactement $k - 2$ monômes disjoints (i.e. en des variables toutes différentes) de degré supérieur ou égal à 3 ; pour tous les annuler il faut au minimum placer un bit de clef nul dans chacun de ces monômes, et par conséquent

$$\mathbb{P}_{\ell<k-2} = 0.$$

Cas introductif : $\ell = k - 2$. Avant de prouver la formule correspondant au cas général, attardons-nous sur le cas particulier plus simple pour lequel $\ell = k - 2$. Cette situation correspond au cas où l'on possède exactement le nombre minimal de bits à zéro nécessaire pour annuler chacun des monômes de degré supérieur ou égal à 3. Pour obtenir la probabilité dans ce cas, il suffit de dénombrer les cas favorables, c'est-à-dire de compter les façons possibles de placer 1 bit nul dans chacun des monômes de degré supérieur ou égal à 3.

Comme nous l'avons vu précédemment, les monômes de degré supérieur ou égal à 3 sont tous positionnés dans la fonction booléenne f_3 et par définition il y a exactement un monôme de chaque degré (du degré 3 au degré k) dans celle-ci. Pour chaque monôme de degré d, on possède d choix possibles pour choisir l'indice de la variable nulle, donc pour annuler l'ensemble des monômes de degré supérieur ou égal à 3 on a $3 \times 4 \times 5 \times \cdots \times k = \frac{k!}{2}$ configurations valides.

8. La version finale de l'article émet une limite plus stricte encore provenant des limites du PRNG : ≪ *Since the permutation generation part of FLIP has only birthday security (with respect to the size of the PRNG), it implies that it is only secure up to* 2^{64} *PRNG outputs when implemented with the AES-128.* ≫.

Pour obtenir la probabilité correspondante, il suffit de faire le ratio de la taille de cet ensemble de cas favorables par le nombre de cas possibles, donné par le nombre total de possibilités pour placer ℓ zéros parmi les N variables que comporte F, soit $\binom{N}{\ell}$:

$$\mathbb{P}_{\ell=k-2} = \frac{k!/2}{\binom{N}{\ell}}.$$

Cas général : $\ell \geq k-2$. Passons à présent au cas général, où l'on s'autorise à deviner plus de $k-2$ positions nulles et ainsi augmenter la probabilité qu'un cycle soit exploitable.

Une première façon de calculer la probabilité de cet événement consiste à procéder comme précédemment et à dénombrer les cas valides, qui correspondent ici à compter de combien de façons différentes on peut placer $i_d \geq 1$ bit(s) nul(s) dans chaque monôme de degré $d \geq 3$ à annuler, sous la condition que les i_j vérifient $\sum_{j=3}^{k} i_j \leq \ell$. Il reste ensuite à placer les $\ell - \sum_{j=3}^{k} i_j$ bits nuls restants dans les autres monômes de degrés inférieur à 3 (en les $N - n_3 + 3$ variables restantes), ce qui nous donne la probabilité finale de :

$$\mathbb{P}_{\ell \geq k-2} = \frac{\sum_{i_3+i_4+\cdots+i_k \leq \ell} \binom{3}{i_3}\binom{4}{i_4}\cdots\binom{k}{i_k}\binom{N-(n_3-3)}{\ell-\sum_{j=3}^{k} i_j}}{\binom{N}{\ell}}.$$

On peut remarquer que lorsque $\ell = k-2$ on retrouve la formule précédente puisqu'on a obligatoirement $i_j = 1 \ \ \forall j \in \{3, \cdots, k\}$:

$$\begin{aligned}
\mathbb{P}_{\ell=k-2} &= \frac{\sum_{i_3+i_4+\cdots+i_k \leq k-2} \binom{3}{i_3}\binom{4}{i_4}\cdots\binom{k}{i_k}\binom{N-(n_3-3)}{\ell-\sum_{j=3}^{k} i_j}}{\binom{N}{\ell}} \\
&= \frac{\binom{3}{1}\binom{4}{1}\cdots\binom{k}{1}\binom{N-(n_3-3)}{\ell-\ell}}{\binom{N}{\ell}} = \frac{3 \times 4 \times \cdots \times k \times 1}{\binom{N}{\ell}} \\
&= \frac{k!/2}{\binom{N}{\ell}}.
\end{aligned}$$

Une autre façon de calculer cette probabilité consiste à s'intéresser à l'événement complémentaire c'est-à-dire aux cas où certains (au moins 1) des monômes de degré supérieur ou égal à 3 ne sont pas annulés. Cette probabilité peut être calculée grâce au principe d'inclusion-exclusion.

Notons D un ensemble d'entiers représentant les degrés des monômes, et par A_D l'événement *la permutation n'envoie aucun des zéros dans les monômes de degré inclus dans D*. Si on note par M_d le monôme (unique) de F de degré d alors A_D est l'événement $\{\forall d \in D, M_d \neq 0\}$. On peut alors exprimer l'événement *le polynôme est de degré supérieur à 3* par :

$$\mathbb{P}\Big(\bigcup_{d \in \{3,\cdots,k\}} A_{\{d\}}\Big) = \sum_{s=1}^{k-2} \left((-1)^s \sum_{\substack{I \subset \{1,\cdots,k-2\} \\ |I|=s}} \mathbb{P}(A_I) \right)$$

ce qui peut être réécrit en :

$$\mathbb{P}(\bigcup_{d\in\{3,\cdots,k\}} A_{\{d\}}) = \frac{\sum_{s=1}^{k-2}\left((-1)^s \sum_{\substack{I\subset\{1,\cdots,k-2\}\\|I|=s}} \binom{N-\sum_{i\notin I} i}{\ell}\right)}{\binom{N}{\ell}}$$

Et la probabilité que nous recherchons est :

$$\mathbb{P}_{\ell\geq k-2} = \mathbb{P}(\bigcap_{d\in\{3,\cdots,k\}} \overline{A_{\{d\}}}) = 1 - \mathbb{P}(\bigcup_{d\in\{3,\cdots,k\}} A_{\{d\}})$$

L'évaluation de ces formules pour les deux instances de FLIP proposées et différentes valeurs de ℓ sont données aux tables 8.3 et 8.4 de la section 8.4.2.

Comme nous allons le voir, ces probabilités sont suffisamment élevées pour pouvoir monter une attaque efficace. Si par exemple on attaque la version à 80 bits de sécurité avec le nombre minimal d'hypothèses ($\ell = 12$), la probabilité qu'un bit de suite chiffrante soit donné par une expression de degré au plus 2 est $\mathbb{P}_{\ell=12} = 2^{-26.335}$. Pour la version à 128 bits de sécurité (et toujours le nombre minimal d'hypothèses) on obtient $\mathbb{P}_{\ell=19} = 2^{-42.382}$.

8.4 Notre attaque

8.4.1 Description

Configuration de départ. On rappelle que l'on considère une attaque à clair connu, c'est-à-dire que l'on suppose avoir accès à C_D bits de la suite chiffrante, que l'on note z_i, où i varie de 0 à $C_D - 1$. Les permutations P_i utilisées à ces instants sont elles aussi connues, donc les données à disposition sont :

$$z_i = F(P_i(k_0, k_1, \cdots, k_{N-1})) \quad \forall i \geq 0$$

où seuls les bits de clef $k_0, \cdots, k_{N-1}$ sont inconnus.

Première étape : hypothèse initiale. La première étape de l'attaque consiste à réaliser une hypothèse sur la position de ℓ des $N/2$ bits de clef nuls, où $\ell \geq k-2$. Cette hypothèse implique qu'il ne nous reste plus que $N - \ell$ bits de clef inconnus et que chaque z_i s'exprime en un maximum de $N - \ell$ variables. La probabilité que cette hypothèse soit correcte est de :

$$\mathbb{P}_{\mathsf{rg}} = \frac{\binom{\frac{N}{2}}{\ell}}{\binom{N}{\ell}}.$$

Cette probabilité est légèrement inférieure à celle que l'on aurait obtenue pour une clef aléatoire[9] (valant $2^{-\ell}$) mais l'avantage est que l'on est assuré qu'il existe au moins une

9. Dans le cas aléatoire, la probabilité que chacun des indices choisis corresponde à un bit de clef nul vaut 2^{-1}, donc la probabilité de choisir ℓ indices corrects est $2^{-\ell}$. Pour le cas où la clef est équilibré, la probabilité que le premier indice soit choisi correctement vaut aussi 2^{-1}, mais cette probabilité diminue pour le second indice puisqu'il n'y a plus que $\frac{N}{2} - 1$ choix favorables parmi les $N - 1$ possibles. La probabilité de choisir correctement ℓ indices dans le cas équilibré vaut donc $\prod_{i=0}^{\ell-1} \frac{\frac{N}{2}-i}{N-i}$. Pour tout $i > 0$ la valeur de $\frac{\frac{N}{2}-i}{N-i}$ est inférieure à 2^{-1}, d'où le résultat.

hypothèse correcte aussi longtemps que $\ell \leq \frac{N}{2}$, ce qui n'est pas le cas avec une clef purement aléatoire pouvant contenir peu de zéros.

Seconde étape : extraction des équations de faible degré. L'objectif de la seconde étape est de rassembler suffisamment d'équations de faible degré pour pouvoir constituer et résoudre un système d'équations en les $N - \ell$ bits de clef inconnus. Pour cela, il faut passer en revue l'ensemble des expressions des C_D bits de suite chiffrante disponibles et en extraire celles pour lesquelles les ℓ bits de clef nuls annulent tous les monômes de degré supérieur à 2. Comme détaillé dans la section précédente, cet événement est de probabilité $\mathbb{P}_\ell$.

Troisième étape : Résolution du système obtenu. La façon la plus simple de résoudre le système quadratique est d'utiliser la technique de linéarisation :

> La linéarisation d'un système d'équations non-linéaires consiste à affecter une nouvelle variable à chacun des monômes de degré supérieur ou égal à 2 présent de sorte à se ramener à un système entièrement linéaire.
>
> Le système obtenu pourra ensuite être résolu par des méthodes simples comme par exemple une élimination de Gauss, qui nécessite un nombre d'opérations de l'ordre de $\mathcal{O}(v^3)$ où v est le nombre de variables présentes dans le système après linéarisation.
>
> Il est important de noter ici que cette méthode accroît le nombre de variables présentes dans le système : dans le cas où une solution unique existe et où on souhaite la déterminer, cela implique qu'il nous faudra autant d'équations que de variables, ce qui peut dans certains cas s'avérer problématique [a]. Si un système d'équations non-linéaires en x variables fait intervenir tous les monômes de degré inférieurs ou égaux à k, il faudra introduire $\sum_{i=2}^{k} \binom{x}{i}$ nouvelles variables (pour donner un ordre de grandeur, pour $k = 5$ et $x = 10$ il faudra introduire pas moins de 627 variables).
>
> *a.* Cela se traduira le plus souvent par une augmentation du nombre de données nécessaires, et par extension en une augmentation du coût de résolution du système.

Comme notre système est quadratique, les seuls termes à linéariser sont les monômes de degré 2. Comme on a un maximum de $\binom{N-\ell}{2}$ tels monômes, le nombre total de variables du système linéarisé sera :

$$v_\ell = N - \ell + \binom{N-\ell}{2}.$$

Si on admet que les équations rassemblées sont aléatoires [10], le nombre d'équations que nous devrons réunir (pour résoudre le système ou trouver une contradiction) sera de l'ordre du nombre d'inconnues. Comme obtenir une équation de faible degré nécessite d'avoir accès à $\mathbb{P}_\ell^{-1}$ bits de la suite chiffrante, cela signifie qu'il faudra en moyenne avoir accès à :

$$C_D = v_\ell \times \frac{1}{\mathbb{P}_\ell}$$

bits de suite chiffrante pour former un système résoluble (et ceci constitue la complexité en données de l'attaque).

10. Nous verrons que nos expérimentations confortent cette hypothèse.

La complexité en temps de l'attaque est dominée par le coût de résolution du système d'équations [11]. Comme cette résolution est effectuée jusqu'à temps d'obtenir une hypothèse correcte sur les ℓ positions de bits nuls de la clef (soit en moyenne $\mathbb{P}_{\mathsf{rg}}$ fois) on a que :

$$C_T = v_\ell^3 \times \frac{1}{\mathbb{P}_{\mathsf{rg}}}$$

Les tables 8.3 et 8.4 données dans la section suivante explicitent les compromis possibles entre les complexités en temps, mémoire et données. On peut y voir qu'augmenter le nombre d'hypothèses initiales ℓ permet de réduire la complexité en données de l'attaque (puisque la probabilité que les monômes de haut degré s'annulent augmente et que par conséquent il faudra observer moins de bits de suite chiffrante pour constituer le système) au prix d'une augmentation de la complexité en temps.

8.4.2 Étude des compromis de complexité possibles

Dans cette section nous étudions les compromis de complexité possibles pour mener notre attaque sur les 2 instances concrètes de FLIP proposées dans [MJSC15]. Pour ces deux versions et pour chacune des valeurs possibles pour ℓ donnant lieu à une attaque valide (i.e. de complexité en temps strictement inférieure à 2^{80} ou 2^{128} selon la version), nous détaillons la probabilité qu'un bit de suite chiffrante soit exploitable ($\mathbb{P}_\ell$), le nombre de variables après linéarisation v_ℓ, la probabilité qu'une hypothèse sur ℓ bits soit juste ($\mathbb{P}_{\mathsf{rg}}$) puis les complexités en temps et en données de l'attaque en résultant.

11. La complexité de la résolution peut être réduite à $v_\ell^{2.8}$ en utilisant l'algorithme de Strassen mais pour simplifier nous considérons ici qu'elle coûte v_ℓ^3.

Version à 80 bits de sécurité (FLIP (47,40,105))

ℓ	$\mathbb{P}_\ell$	v_ℓ	$\mathbb{P}_{\mathsf{rg}}$	C_D	C_T
12	**-26.335**	**-13.992**	**12.528**	**40.326**	**54.503**
13	-23.049	-13.976	13.627	37.025	55.554
14	-20.653	-13.960	14.736	34.613	56.615
15	-18.738	-13.943	15.854	32.682	57.684
16	-17.141	-13.927	16.982	31.069	58.763
17	-15.775	-13.911	18.120	29.686	59.852
18	-14.585	-13.894	19.267	28.480	60.950
19	-13.536	-13.878	20.425	27.414	62.057
20	-12.601	-13.861	21.592	26.462	63.175
21	-11.762	-13.844	22.771	25.606	64.303
22	-11.004	-13.827	23.960	24.831	65.442
23	-10.315	-13.810	25.160	24.125	66.591
24	-9.686	-13.793	26.371	23.479	67.750
25	-9.110	-13.776	27.593	22.886	68.921
26	-8.580	-13.759	28.827	22.339	70.103
27	-8.092	-13.741	30.073	21.833	71.297
28	-7.640	-13.724	31.331	21.364	72.502
29	-7.221	-13.706	32.601	20.927	73.720
30	-6.832	-13.689	33.883	20.520	74.949
31	-6.469	-13.671	35.179	20.140	76.191
32	-6.131	-13.653	36.487	19.784	77.446
33	-5.816	-13.635	37.809	19.450	78.714
34	**-5.520**	**-13.617**	**39.145**	**19.137**	**79.995**
> 34	...	...	...	...	> 80
35	-5.243	-13.598	40.495	18.842	81.290

TABLE 8.3 – $\log_2$ des complexités en temps (C_T) et en données (C_D) de l'attaque en fonction du nombre d'hypothèses initiales (ℓ) pour la version FLIP (47,40,105).

Version à 128 bits de sécurité (**FLIP** (87,82,231))

l	$\mathbb{P}_l$	v_ℓ	$\mathbb{P}_{\mathsf{rg}}$	C_D	C_T
19	**-42.382**	**-16.151**	**19.647**	**58.533**	**68.100**
20	-38.522	-16.144	20.721	54.666	69.151
21	-35.589	-16.136	21.799	51.725	70.206
22	-33.169	-16.128	22.881	49.298	71.266
23	-31.097	-16.121	23.967	47.218	72.329
24	-29.282	-16.113	25.058	45.395	73.397
25	-27.667	-16.105	26.153	43.772	74.469
26	-26.214	-16.098	27.253	42.311	75.546
27	-24.895	-16.090	28.357	40.985	76.627
28	-23.691	-16.082	29.465	39.773	77.712
29	-22.584	-16.074	30.578	38.658	78.802
30	-21.562	-16.067	31.696	37.629	79.896
31	-20.615	-16.059	32.818	36.674	80.994
32	-19.734	-16.051	33.944	35.785	82.097
33	-18.912	-16.043	35.075	34.955	83.205
34	-18.142	-16.035	36.211	34.178	84.317
35	-17.421	-16.027	37.352	33.448	85.434
36	-16.743	-16.020	38.497	32.762	86.556
37	-16.104	-16.012	39.648	32.116	87.683
38	-15.502	-16.004	40.803	31.505	88.814
39	-14.932	-15.996	41.963	30.928	89.950
40	-14.393	-15.988	43.128	30.381	91.091
41	-13.883	-15.980	44.298	29.862	92.237
42	-13.398	-15.972	45.473	29.370	93.388
43	-12.937	-15.964	46.653	28.901	94.543
44	-12.499	-15.956	47.838	28.455	95.704
45	-12.082	-15.947	49.028	28.029	96.870
46	-11.684	-15.939	50.224	27.624	98.042
47	-11.305	-15.931	51.425	27.236	99.218
48	-10.942	-15.923	52.631	26.865	100.400
49	-10.596	-15.915	53.842	26.511	101.586
50	-10.265	-15.907	55.059	26.171	102.779
51	-9.948	-15.898	56.282	25.846	103.976
52	-9.644	-15.890	57.509	25.534	105.180

53	-9.353	-15.882	58.743	25.235	106.388
54	-9.074	-15.873	59.982	24.947	107.602
55	-8.806	-15.865	61.227	24.671	108.822
56	-8.548	-15.857	62.477	24.405	110.048
57	-8.301	-15.848	63.734	24.149	111.279
58	-8.063	-15.840	64.996	23.903	112.516
59	-7.835	-15.831	66.264	23.666	113.758
60	-7.614	-15.823	67.538	23.437	115.007
61	-7.402	-15.815	68.818	23.217	116.262
62	-7.198	-15.806	70.104	23.004	117.522
63	-7.001	-15.797	71.397	22.799	118.789
64	-6.812	-15.789	72.695	22.601	120.062
65	-6.629	-15.780	74.000	22.409	121.341
66	-6.452	-15.772	75.311	22.224	122.627
67	-6.281	-15.763	76.629	22.044	123.918
68	-6.116	-15.754	77.953	21.871	125.216
69	-5.957	-15.746	79.284	21.703	126.521
70	**-5.803**	**-15.737**	**80.621**	**21.540**	**127.832**
> 70	...	...	...	...	> 128
71	-5.655	-15.728	81.965	21.383	129.150

TABLE 8.4: $\log_2$ des complexités en temps (C_T) et en données (C_D) de l'attaque en fonction du nombre d'hypothèses initiales (ℓ pour la version FLIP (87,82,231)).

8.4.3 Vérification expérimentale

Pour confirmer la validité de notre attaque et nos estimations de complexité, nous avons implémenté notre cryptanalyse sur une version réduite du chiffrement. La version considérée possède une clef réduite à $N = 64$ bits ainsi que des valeurs de n_1, n_2 et n_3 réduites respectivement à 14, 14 et 36 (ces valeurs ont été choisies de sorte à conserver les proportions originales des chiffrements FLIP). Nous notons cette version réduite FLIP (14, 14, 36).

Sa fonction de filtrage F possède un degré algébrique de 8 et est définie comme suit :

$$F(x_0, \cdots, x_{63}) = f_1(x_0, \cdots, x_{13}) + f_2(x_{14}, \cdots, x_{27}) + f_3(x_{28}, \cdots, x_{63})$$

où l'ont définit f_1, f_2 et f_3 comme suit :

$$\begin{aligned} f_1(x_0, \cdots, x_{13}) = &\quad L_{14}(x_0, \cdots, x_{13}) &&= x_0 + x_1 + \cdots + x_{13} \\ f_2(x_{14}, \cdots, x_{27}) = &\quad Q_7(x_{14}, \cdots, x_{27}) &&= x_{14}x_{15} + x_{16}x_{17} + \cdots + x_{26}x_{27} \\ f_3(x_{28}, \cdots, x_{63}) = &\quad T_8(x_{28}, \cdots, x_{63}) &&= x_{28} + x_{29}x_{30} + x_{31}x_{32}x_{33} + \cdots + x_{56}x_{57}\cdots x_{63} \end{aligned}$$

Nous avons implémenté cette version réduite et réalisé l'attaque avec $\ell = 8$ hypothèses. Selon les analyses précédentes, les complexités en temps, données et mémoire attendues sont celles données à la table 8.6 et en particulier l'attaque avec $\ell = 8$ aura une complexité en temps totale de l'ordre de $C_T = 2^{40.638}$ opérations élémentaires. La probabilité que l'hypothèse sur les $\ell = 8$ bits de clef nuls soit correcte sera de :

$$\mathbb{P}_{\mathsf{rg}} = 2^{-8.717}$$

et la probabilité que la permutation soit exploitable sera de :

$$\mathbb{P}_{\ell} = 2^{-7.814}.$$

Le système linéaire dépendra de $v_\ell = 1596$ variables. Nos calculs prévoient que $C_D = 2^{18.454}$ bits de suite chiffrante seront nécessaires pour conduire l'attaque.

Les résultats expérimentaux que nous avons obtenus sont donnés à la table 8.5. On peut y voir que la pratique concorde avec la théorie, ce qui confirme la validité de notre modèle.

	Hypothèses	Données	Proportion utilisée	Op. élémentaires	Tps (s.)
Pratique	437.1	$2^{18.455}$	$2^{-7.813}$	$2^{38.588}$	280.93
Théorie	$\mathbb{P}_{\mathsf{rg}}^{-1} = 420.8$	$C_D = 2^{18.454}$	$\mathbb{P}_{\ell} = 2^{-7.814}$	$C_T = 2^{40.638}$	-

TABLE 8.5 – Comparaison des résultats expérimentaux avec les complexités théoriques sur la version réduite FLIP (14,14,36). L'attaque réalisée utilise $\ell = 8$ hypothèses initiales (moyenne obtenue sur 1000 tests réalisés sur un ordinateur de bureau de processeur Intel(R) Xeon(R) CPU W3670 à 3.20GHz (12MB cache) et 8GB de RAM).

ℓ	$\mathbb{P}_\ell$	v_ℓ	$\mathbb{P}_{\mathsf{rg}}$	C_D	C_T	C_M
6	-11.861	10.741	-6.370	22.601	38.592	21.481
7	-9.436	10.691	-7.528	20.126	39.601	21.382
8	-7.814	10.640	-8.717	18.454	40.638	21.280
9	-6.602	10.589	-9.939	17.191	41.706	21.177
10	-5.649	10.536	-11.197	16.185	42.806	21.072
11	-4.876	10.483	-12.493	15.359	43.941	20.966
12	-4.235	10.428	-13.828	14.663	45.113	20.857
13	-3.696	10.373	-15.207	14.069	46.325	20.746
14	-3.237	10.316	-16.631	13.553	47.580	20.633
15	-2.843	10.259	-18.105	13.102	48.881	20.517
16	-2.503	10.200	-19.632	12.703	50.231	20.399
17	-2.208	10.140	-21.217	12.347	51.636	20.279
18	-1.950	10.078	-22.865	12.028	53.100	20.156
19	-1.723	10.015	-24.581	11.739	54.628	20.031
20	-1.524	9.951	-26.373	11.476	56.227	19.903
21	-1.349	9.886	-28.247	11.235	57.904	19.771
22	-1.194	9.819	-30.214	11.013	59.670	19.637
23	-1.057	9.750	-32.284	10.807	61.534	19.500
24	**-0.935**	**9.679**	**-34.472**	**10.615**	**63.510**	**19.359**
> 24	...	...	...	...	> 64	...
25	-0.827	9.607	-36.794	10.435	65.616	19.215

TABLE 8.6 – $\log_2$ des complexités en temps (C_T) et en données (C_D) de l'attaque en fonction du nombre d'hypothèses initiales (ℓ) pour notre version réduite FLIP (14, 14, 36).

Bien que les équations obtenues aient une structure très spécifique, nous avons pu remarquer qu'elles se comportaient de façon similaire à des équations aléatoires : nous avons en effet remarqué que c'est uniquement après avoir généré environ 1590 équations que la première équation dépendante des précédentes était trouvée, ce qui va dans le sens de ce qui est énoncé par exemple dans [LN83].

8.4.4 Discussion et améliorations possibles

1. **Réduction de la complexité en données.** Il est possible de réduire un peu plus la complexité en données de l'attaque en tirant parti du fait que la graine du PRNG est publique. En effet ce point implique que les expressions de tous les bits de suite chiffrante à venir sont connus, ce qui permet à l'attaquant de réaliser les ℓ hypothèses de positions de bits de clef nuls en conséquence. Il choisira les indices de sorte que beaucoup des premières permutations soient exploitables, et qu'au final il soit nécessaire d'avoir accès à moins de bits de suite chiffrante pour constituer le système que dans le cas aléatoire.

2. **Réalisation de pré-calculs.** Le facteur dominant la complexité en temps de l'attaque provient de la résolution des systèmes linéaires. Ces calculs ne dépendant que des permutations (connues) et des hypothèses (choisies par l'attaquant), il est possible de réaliser les inversions des systèmes dans une étape de pré-calcul, de sorte qu'il ne reste plus ensuite qu'à insérer la valeur des bits de suite chiffrante pour obtenir le résultat. L'inconvénient de cette technique réside dans l'augmentation significative de la complexité en mémoire nécessaire.

3. **Indépendance de l'attaque au changement de graine.** Notre attaque a la propriété d'être indépendante à un changement de graine en cours de chiffrement. Si le système est réinitialisé, l'attaquant pourra continuer à collecter des équations et construire un système avec des équations obtenues sous différentes graines. Cela provient du fait que tant que la clef n'est pas modifiée les bits de la suite chiffrante s'expriment comme une équation en les mêmes inconnues.

4. **Sécurité.** Notre attaque indique que le niveau de sécurité de la famille de chiffrements FLIP est de l'ordre de $\sqrt{N}$, où N est la taille de la clef. Nous avons vu que la complexité en temps de nos attaques est égale à :

$$C_T = v_\ell^3 \times \frac{1}{\mathbb{P}_{\mathsf{rg}}}.$$

Comme $\ell \ll N$, $\mathbb{P}_{\mathsf{rg}}$ est environ équivalent à $2^{-\ell}$. Aussi, comme $v_\ell = N - \ell + \binom{N-\ell}{2}$, nous obtenons l'approximation de C_T suivante :

$$C_T \sim N^6 \times 2^\ell.$$

On sait de plus que le nombre d'hypothèses ℓ minimum nécessaire pour conduire l'attaque est égal au nombre de monômes de degré supérieur ou égal à 3 dans T_{n_3}, ce qui implique que $n_3 = (\ell+2)(\ell+3)/2$ et par suite que $\ell \sim \sqrt{n_3}$. On obtient le résultat annoncé :

$$\log C_T \sim \alpha\sqrt{N}.$$

La figure 8.4 appuie cette théorie : elle représente l'évolution de la complexité en temps de l'attaque en fonction de la taille de la clef pour des instances de FLIP de la forme FLIP (n_1, n_2, n_3) où $N = n_1 + n_2 + n_3 = 2n_3$ (ce qui est cohérent avec les paramètres proposés dans [MJSC15]). Le paramètre ℓ est choisi comme le nombre minimum d'hypothèses nécessaires pour effectuer l'attaque c'est-à-dire $\ell = k - 2$.

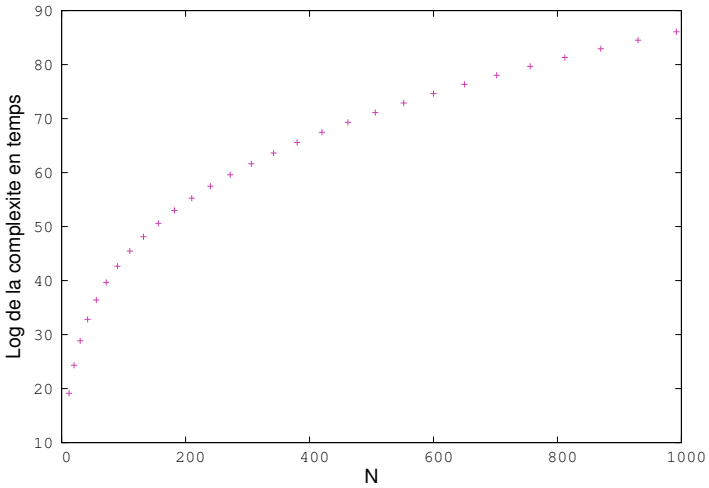


FIGURE 8.4 – Évolution de la complexité en temps en fonction de la taille de la clef N.

8.5 Correction réalisée par les auteurs

Comme nous l'avons dit en introduction de ce chapitre, notre travail a permis de mettre en avant une faiblesse des premières instances de FLIP face aux attaques de type *guess-and-determine*. Nous avons rapidement averti les auteurs de ces vulnérabilités ce qui leur a permis de proposer une nouvelle série de paramètres assurant la résistance aux attaques précédemment considérées (attaques algébriques, algébriques rapides, par corrélation, etc) et aux attaques utilisant nos techniques.

Pour se faire, les concepteurs apportèrent deux modifications à leur schéma :
— À sécurité équivalente, la taille de l'état interne (et donc de la clef) est augmentée (elle est environ multipliée par un facteur 3).
— La fonction booléenne de filtrage F est modifiée pour faire intervenir plus de monômes de haut degré.

La structure et l'idée globale du chiffrement sont néanmoins conservées : F est toujours définie comme une fonction booléenne en N variables (les N bits de la clef) et est formée par la somme directe de trois fonctions booléennes f_1, f_2 et f_3 en respectivement n_1, n_2 et n_3 variables. Si les définitions de f_1 et de f_2 ne changent pas, f_3 est redéfinie de façon plus générale :
— $f_1(x_0, \cdots, x_{n_1-1}) = L_{n_1}$,
— $f_2(x_{n_1}, \cdots, x_{n_1+n_2-1}) = Q_{n_2/2}$,
— $f_3(x_{n_1+n_2}, \cdots, x_{n_1+n_2+n_3-1})$ est donnée par la somme directe de nb fonctions T_k de *type* T (de sorte que chaque fonction de *type* T agisse sur des variables différentes) notées ${}^{nb}\Delta^k$ (nous reprenons les notations utilisées dans [MJSC16]).

F est donc donnée par :

$$\begin{array}{rccc} F: & \mathbb{F}_2^{n_1+n_2+n_3-1} & \longrightarrow & \mathbb{F}_2 \\ & x_0, \cdots, x_{n_1+n_2+n_3-1} & \longmapsto & L_{n_1} \oplus Q_{n_2/2} \oplus \bigoplus_{i=1}^{nb} T_k. \end{array}$$

L'analyse de sécurité réalisée par les auteurs (nous renvoyons à [MJSC16]) a conduit à proposer 4 nouvelles instances. Celles-ci sont notées FLIP $(n_1, n_2, {}^{nb}\Delta^k)$ où n_1 et n_2 sont respectivement le nombre de variables de f_1 et f_2 et où ${}^{nb}\Delta^k$ indique que f_3 est composée de nb fonctions de *type* T de degré k. Les deux premières instances correspondent à une sécurité de 80 bits et sont données par :

— FLIP $(42, 128, {}^{8}\Delta^9)$, avec $N = 530$
— FLIP $(46, 136, {}^{4}\Delta^{15})$, avec $N = 662$

Les deux instances correspondant à une sécurité de 128 bits sont :

— FLIP $(82, 224, {}^{8}\Delta^{16})$, avec $N = 1384$
— FLIP $(86, 238, {}^{5}\Delta^{23})$, avec $N = 1704$.

Les clefs utilisées sont environ 3 fois plus grandes pour ces instances que pour celles que nous avons attaquées et possèdent plus de monômes de degré supérieur ou égal à 3. Ces deux propriétés permettent de mettre en échec notre cryptanalyse. Pour ne donner qu'un exemple, FLIP $(42, 128, {}^{8}\Delta^9)$ possède 56 monômes de degré supérieur ou égal à 3 (au lieu de 12 dans la précédente version à 80 bits de sécurité). En plus de devoir réaliser plus d'hypothèses sur des positions nulles et de considérer au minimum $\ell = 56$, l'attaquant aura à chaque essai une chance moindre que ses hypothèses soient correctes (cela provient du fait que la clef est beaucoup plus grande).

Les auteurs évoquent aussi la possibilité de considérer des *sommes indirectes*. Ce changement conduirait à un schéma plus difficile à analyser (la transmission des bonnes propriétés des fonctions f_i à la suite chiffrante n'est plus aussi simple que dans le cas des sommes directes) mais devrait permettre de réduire la taille de l'état interne.

Une possibilité de mise en œuvre de cette idée consisterait à calculer b instances de FLIP en parallèle (chacune avec la même fonction de filtrage F mais avec des permutations différentes), puis à additionner les b bits obtenus pour obtenir le bit de suite chiffrante. Une telle construction est notée b-FLIP. Ils suggèrent l'instance 10-FLIP $(10, 20, {}^{1}\Delta^{20})$ pour une sécurité de 80 bits et l'instance 15-FLIP $(15, 30, {}^{1}\Delta^{30})$ pour une sécurité de 128 bits.

8.6 Conclusion

Dans ce chapitre nous avons décrit une attaque de type *guess-and-determine* sur une des premières propositions de chiffrement symétrique visant une intégration efficace dans un schéma de chiffrement utilisant un chiffrement complètement homomorphe hybride. Notre attaque exploite le faible nombre de monômes de haut degré de la fonction de filtrage F pour obtenir un système d'équations quadratiques facile à résoudre. Nos commentaires ont permis aux auteurs de réévaluer leurs jeux de paramètres et de proposer une version de FLIP résistante à notre attaque. Bien que la taille de clef ait été multipliée par 3, leurs nouvelles instances restent très compétitives et adaptées à une intégration dans un schéma hybride pour le chiffrement homomorphe.

L'originalité de cette construction impose que plusieurs analyses de sécurité externes soient réalisées pour confirmer sa sécurité.

Conclusion

Dans ce manuscrit, nous avons énoncé plusieurs résultats de cryptanalyse relatifs aux deux grandes familles de chiffrement symétrique, à savoir les chiffrements par blocs et les chiffrements à flot. Contrairement à la majorité des cryptanalyses parues récemment, qui portent sur des versions réduites des chiffrements, les 5 cryptanalyses présentées ici attaquent les versions complètes des algorithmes et montrent que les arguments de sécurité de leurs concepteurs sont faux.

Les attaques présentées dans la première partie portent sur 3 chiffrements par blocs récents et utilisent des techniques différentielles. Nos résultats mettent en avant le fait que, bien que la cryptanalyse différentielle ait été l'une des premières attaques découvertes — et donc aussi l'une des plus étudiées —, il reste difficile d'évaluer avec justesse la résistance d'un schéma à cette attaque au moment de sa conception.

Ces cryptanalyses sont aussi une illustration probante de la complexité du travail du concepteur, qui doit savoir concilier sécurité et adéquation avec les applications visées. En effet, nous avons vu que notre attaque sur le chiffrement KLEIN reposait sur la faible diffusion de l'opération *MixColumns*, choisie par les concepteurs parce qu'elle permettait des bonnes performances. Dans une moindre mesure, les faiblesses exhibées dans le chiffrement PICARO sont aussi liées aux choix faits pour répondre aux exigences de l'application visée — ici là conservation de bonnes performances lors de l'utilisation d'un schéma de masquage. L'exemple le plus éloquent provient sans doute du chiffrement `Zorro` : pour satisfaire aux contraintes du schéma de masquage, les auteurs se sont éloignés de la structure de SPN classique et ont considéré des couches non-linéaires partielles. Leur construction s'éloignait alors trop des chiffrements usuels pour pouvoir être étudiée avec les outils existants. Nos travaux ont résolu ce problème en proposant les algorithmes nécessaires et ont pu répondre à la question laissée ouverte par les recherches précédentes, qui était de savoir s'il existait une variante de `Zorro` résistant aux attaques différentielles de base.

Dans la second partie de ce manuscrit nous avons décrit des attaques sur les instanciations concrètes de 2 nouvelles constructions de chiffrement à flot nommées Sprout et FLIP. Nos recherches ont mis en évidence des problèmes dans les deux chiffrements : la faiblesse de Sprout provient de l'intervention non-linéaire de la clef dans la mise à jour des registres du chiffrement ainsi que de la faible dépendance entre les différentes parties de l'état interne. Nous avons montré que les premiers paramètres proposés pour FLIP sont insuffisants car ils permettent de monter une attaque de type *guess-and-determine* : là encore l'application visée (l'intégration dans un schéma de FHE hybride) a poussé les auteurs à s'éloigner des constructions habituelles de chiffrement à flot, et malheureusement la sécurité obtenue s'est avérée trop faible. Nos recherches ont permis d'orienter les concepteurs vers des choix de paramètres plus sûrs.

Nos travaux soulèvent la question de la compréhension précise de l'ensemble des choix à disposition des concepteurs de chiffrements par blocs et à flot. Un des aspects qu'il serait intéressant d'étudier est le niveau minimum de non-linéarité (au sens large) qu'un chiffrement peut atteindre tout en restant sûr. Cette problématique est apparue avec les chiffrements dédiés aux schémas hybrides pour le FHE (pour lesquels il faut minimiser la profondeur multiplicative) et ceux s'intéressant à la facilité du masquage (pour lesquels il faut minimiser le nombre d'opérations non-linéaires). Eu égard au faible nombre de primitives sûres répondant à ces problèmes, cette question apparaît loin d'être triviale.

Annexes

Annexe A

Preuve des propriétés utilisées dans l'attaque sur KLEIN

A.1 Propriétés déduites du développement de l'opération *MixColumns*

La majeure partie des propriétés utilisées dans notre attaque sur KLEIN provient des caractéristiques de l'opération linéaire *MixColumns*. Nous donnons donc ici le développement de son calcul ainsi que de son inverse, duquel elles se déduisent facilement.

Comme précédemment, on note le développement binaire d'un octet a par :

$$a = (a_7, a_6, a_5, a_4 | a_3, a_2, a_1, a_0)$$

où a_0 est le bits de poids faible, a_7 le bit de poids fort. Dans toute la suite, les bits du quartet bas seront notés en rouge et ceux du quartet haut seront notés en bleu.

Le développement du calcul de *MixColumns* donne :

$$\begin{pmatrix} e \\ f \\ g \\ h \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \times \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} \tag{A.1}$$

$$\begin{aligned}
e_7 &= a_6 + b_7 + b_6 + c_7 + d_7 \\
e_6 &= a_5 + b_6 + b_5 + c_6 + d_6 \\
e_5 &= a_4 + b_5 + b_4 + c_5 + d_5 \\
e_4 &= a_7 + a_3 + b_7 + b_4 + b_3 + c_4 + d_4 \\
e_3 &= a_7 + a_2 + b_7 + b_2 + b_3 + c_3 + d_3 \\
e_2 &= a_1 + b_2 + b_1 + c_2 + d_2 \\
e_1 &= a_7 + a_0 + b_7 + b_1 + b_0 + c_1 + d_1 \\
e_0 &= a_7 + b_7 + b_0 + c_0 + d_0
\end{aligned}$$

$$\begin{aligned}
f_7 &= a_7 + b_6 + c_7 + c_6 + d_7 \\
f_6 &= a_6 + b_5 + c_6 + c_5 + d_6 \\
f_5 &= a_5 + b_4 + c_5 + c_4 + d_5 \\
f_4 &= a_4 + b_7 + b_3 + c_7 + c_4 + c_3 + d_4 \\
f_3 &= a_3 + b_2 + b_7 + c_7 + c_3 + c_2 + d_3 \\
f_2 &= a_2 + b_1 + c_2 + c_1 + d_2 \\
f_1 &= a_1 + b_7 + b_0 + c_7 + c_1 + c_0 + d_1 \\
f_0 &= a_0 + b_7 + c_7 + c_0 + d_0
\end{aligned}$$

$$
\begin{aligned}
g_7 &= a_7 + b_7 + c_6 + d_6 + d_7 \\
g_6 &= a_6 + b_6 + c_5 + d_6 + d_5 \\
g_5 &= a_5 + b_5 + c_4 + d_5 + d_4 \\
g_4 &= a_4 + b_4 + c_7 + c_3 + d_7 + d_4 + d_3 \\
g_3 &= a_3 + b_3 + c_7 + c_2 + d_7 + d_3 + d_2 \\
g_2 &= a_2 + b_2 + c_1 + d_2 + d_1 \\
g_1 &= a_1 + b_1 + c_7 + c_0 + d_7 + d_1 + d_0 \\
g_0 &= a_0 + b_0 + c_7 + d_7 + d_0
\end{aligned}
\qquad
\begin{aligned}
h_7 &= a_7 + a_6 + b_7 + c_7 + d_6 \\
h_6 &= a_6 + a_5 + b_6 + c_6 + d_5 \\
h_5 &= a_5 + a_4 + b_5 + c_5 + d_4 \\
h_4 &= a_7 + a_4 + a_3 + b_4 + c_4 + d_7 + d_3 \\
h_3 &= a_7 + a_3 + a_2 + b_3 + c_3 + d_7 + d_2 \\
h_2 &= a_2 + a_1 + b_2 + c_2 + d_1 \\
h_1 &= a_7 + a_1 + a_0 + b_1 + c_1 + d_7 + d_0 \\
h_0 &= a_7 + a_0 + b_0 + c_0 + d_7
\end{aligned}
$$

Propriété. *Soit* $\mathtt{X} \in \mathbb{F}_2^4$ *un quartet de valeur quelconque. Si la différence de 32 bits en entrée de MixColumns est de la forme* `0X0X0X0X` *alors la différence de sortie sera de la même forme avec probabilité* 2^{-3}.

Démonstration. Pour prouver ceci, on considère que l'entrée (a, b, c, d) possède ses quartets hauts inactifs. Les équations ci-dessus indiquent qu'annuler les quartets hauts de la sortie (e, f, g, h) force à vérifier :

$$
\begin{cases}
a_3 + b_3 = 0 \\
b_3 + c_3 = 0 \\
c_3 + d_3 = 0 \\
d_3 + a_3 = 0
\end{cases}
$$

et, comme seulement 3 d'entre elles sont indépendantes (la quatrième équation est la somme des 3 premières), la probabilité de cet événement est 2^{-3}. □

L'analogue de cette propriété pour les quartets bas inactifs est la suivante :

Propriété. *Soit* $\mathtt{X} \in \mathbb{F}_2^4$ *un quartet de valeur quelconque. Si la différence de 32 bits en entrée de MixColumns est de la forme* `X0X0X0X0` *alors la différence de sortie sera de la même forme avec probabilité* 2^{-3}.

Démonstration. Elle se démontre en utilisant le développement précédent et en considérant que l'entrée (a, b, c, d) a ses quartets bas inactifs (donc tous les bits d'indices 0 à 3 sont nuls) : obtenir une sortie pour laquelle les quartets bas sont inactifs correspond à satisfaire :

$$
\begin{cases}
a_7 + b_7 = 0 \\
b_7 + c_7 = 0 \\
c_7 + d_7 = 0 \\
d_7 + a_7 = 0
\end{cases}
$$

Seules 3 équations sont indépendantes donc, là encore, la probabilité de l'événement est bien 2^{-3}.

□

Une autre propriété intéressante liée à la précédente indique comment calculer les quartets bas de sortie de l'opération *MixColumns* lorsque uniquement les quartets bas de l'entrée sont connus. De façon surprenante, uniquement 3 bits d'information sont nécessaires :

Propriété. *Pour calculer les 4 quartets bas en sortie de l'opération MixColumns en connaissant les quartets bas de son entrée, il est nécessaire de connaître la valeur des 3 bits suivants provenant des positions hautes :*

$$\begin{cases} a_7 + b_7 \\ b_7 + c_7 \\ c_7 + d_7 \end{cases}$$

Son homologue sur les quartets hauts est la suivante :

Propriété. *Pour calculer les 4 quartets hauts en sortie de l'opération MixColumns en connaissant les quartets hauts de son entrée, il est nécessaire de connaître la valeur des 3 bits suivants provenant des positions basses :*

$$\begin{cases} a_3 + b_3 \\ b_3 + c_3 \\ c_3 + d_3 \end{cases}$$

A.2 Propriétés déduites du développement de l'inverse de l'opération *MixColumns*

L'inverse de l'opération *MixColumns* s'obtient par le calcul suivant :

$$\begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{pmatrix} \times \begin{pmatrix} e \\ f \\ g \\ h \end{pmatrix} \tag{A.2}$$

Après développement on obtient les formules suivantes :

$$\begin{aligned}
a_7 &= e_6 + e_5 + e_4 + f_7 + f_6 + f_4 + g_7 + g_5 + g_4 + h_7 + h_4 \\
a_6 &= e_7 + e_5 + e_4 + e_3 + f_7 + f_6 + f_5 + f_3 + g_7 + g_6 + g_4 + g_3 + h_7 + h_6 + h_3 \\
a_5 &= e_6 + e_4 + e_3 + e_2 + f_7 + f_6 + f_5 + f_4 + f_2 + g_6 + g_5 + g_3 + g_2 + h_7 + h_6 + h_5 + h_2 \\
a_4 &= e_5 + e_3 + e_2 + e_1 + f_7 + f_6 + f_5 + f_4 + f_3 + f_1 + g_7 + g_5 + g_4 + g_2 + g_1 + h_6 + h_5 + h_4 + h_1 \\
a_3 &= e_6 + e_5 + e_2 + e_1 + e_0 + f_5 + f_3 + f_2 + f_0 + g_7 + g_6 + g_5 + g_3 + g_1 + g_0 + h_7 + h_5 + h_3 + h_0 \\
a_2 &= e_6 + e_1 + e_0 + f_7 + f_6 + f_2 + f_1 + g_6 + g_2 + g_0 + h_7 + h_6 + h_2 \\
a_1 &= e_5 + e_0 + f_7 + f_6 + f_5 + f_1 + f_0 + g_7 + g_5 + g_1 + h_6 + h_5 + h_1 \\
a_0 &= e_7 + e_6 + e_5 + f_7 + f_5 + f_0 + g_6 + g_5 + g_0 + h_5 + h_0
\end{aligned}$$

$$\begin{aligned}
b_7 &= e_7 + e_4 + f_6 + f_5 + f_4 + g_7 + g_6 + g_4 + h_7 + h_5 + h_4 \\
b_6 &= e_7 + e_6 + e_3 + f_7 + f_5 + f_4 + f_3 + g_7 + g_6 + g_5 + g_3 + h_7 + h_6 + h_4 + h_3 \\
b_5 &= e_7 + e_6 + e_5 + e_2 + f_6 + f_4 + f_3 + f_2 + g_7 + g_6 + g_5 + g_4 + g_2 + h_6 + h_5 + h_3 + h_2 \\
b_4 &= e_6 + e_5 + e_4 + e_1 + f_5 + f_3 + f_2 + f_1 + g_7 + g_6 + g_5 + g_4 + g_3 + g_1 + h_7 + h_5 + h_4 + h_2 + h_1 \\
b_3 &= e_7 + e_5 + e_3 + e_0 + f_6 + f_5 + f_2 + f_1 + f_0 + g_5 + g_3 + g_2 + g_0 + h_7 + h_6 + h_5 + h_3 + h_1 + h_0 \\
b_2 &= e_7 + e_6 + e_2 + f_6 + f_1 + f_0 + g_7 + g_6 + g_2 + g_1 + h_6 + h_2 + h_0 \\
b_1 &= e_6 + e_5 + e_1 + f_5 + f_0 + g_7 + g_6 + g_5 + g_1 + g_0 + h_7 + h_5 + h_1 \\
b_0 &= e_5 + e_0 + f_7 + f_6 + f_5 + g_7 + g_5 + g_0 + h_6 + h_5 + h_0
\end{aligned}$$

$$
\begin{aligned}
c_7 &= e_7 + e_5 + e_4 + f_7 + f_4 + g_6 + g_5 + g_4 + h_7 + h_6 + h_4 \\
c_6 &= e_7 + e_6 + e_4 + e_3 + f_7 + f_6 + f_3 + g_7 + g_5 + g_4 + g_3 + h_7 + h_6 + h_5 + h_3 \\
c_5 &= e_6 + e_5 + e_3 + e_2 + f_7 + f_6 + f_5 + f_2 + g_6 + g_4 + g_3 + g_2 + h_7 + h_6 + h_5 + h_4 + h_2 \\
c_4 &= e_7 + e_5 + e_4 + e_2 + e_1 + f_6 + f_5 + f_4 + f_1 + g_5 + g_3 + g_2 + g_1 + h_7 + h_6 + h_5 + h_4 + h_3 + h_1 \\
c_3 &= e_7 + e_6 + e_5 + e_3 + e_1 + e_0 + f_7 + f_5 + f_3 + f_0 + g_6 + g_5 + g_2 + g_1 + g_0 + h_5 + h_3 + h_2 + h_0 \\
c_2 &= e_6 + e_2 + e_0 + f_7 + f_6 + f_2 + g_6 + g_1 + g_0 + h_7 + h_6 + h_2 + h_1 \\
c_1 &= e_7 + e_5 + e_1 + f_6 + f_5 + f_1 + g_5 + g_0 + h_7 + h_6 + h_5 + h_1 + h_0 \\
c_0 &= e_6 + e_5 + e_0 + f_5 + f_0 + g_7 + g_6 + g_5 + h_7 + h_5 + h_0
\end{aligned}
$$

$$
\begin{aligned}
d_7 &= e_7 + e_6 + e_4 + f_7 + f_5 + f_4 + g_7 + g_4 + h_6 + h_5 + h_4 \\
d_6 &= e_7 + e_6 + e_5 + e_3 + f_7 + f_6 + f_4 + f_3 + g_7 + g_6 + g_3 + h_7 + h_5 + h_4 + h_3 \\
d_5 &= e_7 + e_6 + e_5 + e_4 + e_2 + f_6 + f_5 + f_3 + f_2 + g_7 + g_6 + g_5 + g_2 + h_6 + h_4 + h_3 + h_2 \\
d_4 &= e_7 + e_6 + e_5 + e_4 + e_3 + e_1 + f_7 + f_5 + f_4 + f_2 + f_1 + g_6 + g_5 + g_4 + g_1 + h_5 + h_3 + h_2 + h_1 \\
d_3 &= e_5 + e_3 + e_2 + e_0 + f_7 + f_6 + f_5 + f_3 + f_1 + f_0 + g_7 + g_5 + g_3 + g_0 + h_6 + h_5 + h_2 + h_1 + h_0 \\
d_2 &= e_7 + e_6 + e_2 + e_1 + f_6 + f_2 + f_0 + g_7 + g_6 + g_2 + h_6 + h_1 + h_0 \\
d_1 &= e_7 + e_6 + e_5 + e_1 + e_0 + f_7 + f_5 + f_1 + g_6 + g_5 + g_1 + h_5 + h_0 \\
d_0 &= e_7 + e_5 + e_0 + f_6 + f_5 + f_0 + g_5 + g_0 + h_7 + h_6 + h_5
\end{aligned}
$$

Ces équations nous permettent d'établir des propriétés similaires à celles établies pour *MixColumns* :

Propriété A.1. *Soit* $\mathtt{X} \in \mathbb{F}_2^4$ *un quartet de valeur quelconque. Si la différence de 32 bits en sortie de MixColumns est de la forme* `OXOXOXOX` *alors la différence en entrée de l'opération sera de la même forme avec probabilité* 2^{-3}.

Démonstration. En effet, si les quartets hauts de (e, f, g, h) sont tous nuls l'expression des quartets hauts de (a, b, c, d) devient :

$$
\begin{aligned}
a_7 &= 0 \\
a_6 &= e_3 + f_3 + g_3 + h_3 \\
a_5 &= e_3 + e_2 + f_2 + g_3 + g_2 + h_2 \\
a_4 &= e_3 + e_2 + e_1 + f_3 + f_1 + g_2 + g_1 + h_1 \\
b_7 &= 0 \\
b_6 &= e_3 + f_3 + g_3 + h_3 \\
b_5 &= e_2 + f_3 + f_2 + g_2 + h_3 + h_2 \\
b_4 &= e_1 + f_3 + f_2 + f_1 + g_3 + g_1 + h_2 + h_1 \\
c_7 &= 0 \\
c_6 &= e_3 + f_3 + g_3 + h_3 \\
c_5 &= e_3 + e_2 + f_2 + g_3 + g_2 + h_2 \\
c_4 &= e_2 + e_1 + f_1 + g_3 + g_2 + g_1 h_3 + h_1 \\
d_7 &= 0 \\
d_6 &= e_3 + f_3 + g_3 + h_3 \\
d_5 &= e_2 + f_3 + f_2 + g_2 + h_3 + h_2 \\
d_4 &= e_3 + e_1 + f_2 + f_1 + g_1 + h_3 + h_2 + h_1.
\end{aligned}
$$

On voit facilement que ces quantités deviennent toutes nulles sous uniquement 3 conditions (linéairement indépendantes) :

$$
\begin{cases}
e_3 + f_3 + g_3 + h_3 = 0 \\
e_3 + e_2 + f_2 + g_3 + g_2 + h_2 = 0 \\
e_3 + e_2 + e_1 + f_3 + f_1 + g_2 + g_1 + h_1 = 0
\end{cases}
$$

Ce qui prouve la propriété. □

On a une propriété analogue pour les quartets hauts :

Propriété. *Soit* $X \in \mathbb{F}_2^4$ *un quartet de valeur quelconque. Si la différence de 32 bits en sortie de MixColumns est de la forme* `XOXOXOXO` *alors la différence en entrée de l'opération sera de la même forme avec probabilité* 2^{-3}.

Démonstration. En effet l'expression des quartets bas devient :

$$\begin{aligned}
a_3 &= e_6 + e_5 + f_5 + g_7 + g_6 + g_5 + h_7 + h_5 \\
a_2 &= e_6 + f_7 + f_6 + g_6 + h_7 + h_6 \\
a_1 &= e_5 + f_7 + f_6 + f_5 + g_7 + g_5 + h_6 + h_5 \\
a_0 &= e_7 + e_6 + e_5 + f_7 + f_5 + g_6 + g_5 + h_5 \\
b_3 &= e_7 + e_5 + f_6 + f_5 + g_5 + h_7 + h_6 + h_5 \\
b_2 &= e_7 + e_6 + f_6 + g_7 + g_6 + h_6 \\
b_1 &= e_6 + e_5 + f_5 + g_7 + g_6 + g_5 + h_7 + h_5 \\
b_0 &= e_5 + f_7 + f_6 + f_5 + g_7 + g_5 + h_6 + h_5 \\
c_3 &= e_7 + e_6 + e_5 + f_7 + f_5 + g_6 + g_5 + h_5 \\
c_2 &= e_6 + f_7 + f_6 + g_6 + h_7 + h_6 \\
c_1 &= e_7 + e_5 + f_6 + f_5 + g_5 + h_7 + h_6 + h_5 \\
c_0 &= e_6 + e_5 + f_5 + g_7 + g_6 + g_5 + h_7 + h_5 \\
d_3 &= e_5 + f_7 + f_6 + f_5 + g_7 + g_5 + h_6 + h_5 \\
d_2 &= e_7 + e_6 + f_6 + g_7 + g_6 + h_6 \\
d_1 &= e_7 + e_6 + e_5 + f_7 + f_5 + g_6 + g_5 + h_5 \\
d_0 &= e_7 + e_5 + f_6 + f_5 + g_5 + h_7 + h_6 + h_5
\end{aligned} \tag{A.3}$$

Et ceux-ci s'annulent sous 3 conditions linéairement indépendantes, par exemple :

$$\begin{cases}
e_6 + e_5 + f_5 + g_7 + g_6 + g_5 + h_7 + h_5 = 0 \\
e_6 + f_7 + f_6 + g_6 + h_7 + h_6 = 0 \\
e_7 + e_6 + e_5 + f_7 + f_5 + g_6 + g_5 + h_5 = 0
\end{cases}$$

ce qui prouve la propriété. □

Ces développements permettent aussi de démontrer la propriété 3.5 que nous redonnons ici :

Propriété. *Pour obtenir les quartets bas résultant de l'inversion de l'opération MixColumns connaissant les quartets bas de l'entrée (e, f, g, h), il est nécessaire de connaître 3 valeurs dépendant des quartets hauts inconnus :*

$$\begin{cases}
e_6 + e_5 + f_5 + g_7 + g_6 + g_5 + h_7 + h_5 \\
e_6 + f_7 + f_6 + g_6 + h_7 + h_6 \\
e_7 + e_6 + e_5 + f_7 + f_5 + g_6 + g_5 + h_5.
\end{cases}$$

Démonstration. En effet, si on connaît uniquement la valeur des quartets bas de l'entrée (e, f, g, h), il nous manque les 16 bits exprimés en A.3 pour calculer la valeur exacte des quartets bas de (a, b, c, d). Comme nous venons de le voir, il suffit de faire une hypothèse sur 3 bits pour en déduire les 13 autres et calculer l'ensemble des quartets bas de (a, b, c, d).

□

De la même manière on prouve la propriété suivante :

Propriété. *Pour obtenir les quartets hauts résultant de l'inversion de l'opération MixColumns connaissant les quartets hauts de l'entrée (e, f, g, h), il est nécessaire de connaître 3 valeurs dépendant des quartets bas inconnus :*

$$\begin{cases} e_3 + f_3 + g_3 + h_3 \\ e_3 + e_2 + f_2 + g_3 + g_2 + h_2 \\ e_3 + e_2 + e_1 + f_3 + f_1 + g_2 + g_1 + h_1. \end{cases}$$

Annexe B

Quelques remarques sur la boîte-S de PICARO

On donne ici quelques observations sur la boîte-S de PICARO, définie par :

$$\begin{array}{rcl} S : \mathbb{F}_{2^4} \times \mathbb{F}_{2^4} & \to & \mathbb{F}_{2^4} \times \mathbb{F}_{2^4} \\ (x, y) & \mapsto & (xy, (x^3 + \texttt{0x02})(y^3 + \texttt{0x04})) \end{array}$$

Propriété B.1. *S possède 86 images possibles, et chacune (à l'exception de* `08` *qui est l'image de* `00`*) possède exactement 3 antécédents.*

Propriété B.2. *Si on s'intéresse uniquement aux quartets bas des valeurs de la table de correspondance de S on peut voir que chaque colonne (resp. chaque ligne) est répétée 3 fois.*

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	08	0c	03	06	06	04	05	06	05	04	0c	0c	04	03	05	03
10	0a	1f	29	3b	4b	55	62	7b	82	95	af	bf	c5	d9	e2	f9
20	01	2d	45	6a	8a	ac	cf	ea	9f	bc	dd	fd	1c	35	5f	75
30	0f	34	61	52	c2	fb	a3	92	13	2b	74	44	db	e1	b3	81
40	0f	44	81	c2	92	db	13	52	b3	fb	34	74	2b	61	a3	e1
50	0e	59	a4	f8	d8	87	7c	28	3c	67	99	c9	e7	b4	4c	14
60	02	63	ca	ad	1d	71	d7	bd	27	41	e3	83	31	5a	f7	9a
70	0f	74	e1	92	52	2b	b3	c2	a3	db	44	34	fb	81	13	61
80	02	83	9a	1d	bd	31	27	ad	f7	71	63	e3	41	ca	d7	5a
90	0e	99	b4	28	f8	67	4c	d8	7c	e7	c9	59	87	14	3c	a4
a0	0a	af	d9	7b	3b	95	e2	4b	62	c5	bf	1f	55	f9	82	29
b0	0a	bf	f9	4b	7b	c5	82	3b	e2	55	1f	af	95	29	62	d9
c0	0e	c9	14	d8	28	e7	3c	f8	4c	87	59	99	67	a4	7c	b4
d0	01	dd	35	ea	6a	bc	5f	8a	cf	1c	fd	2d	ac	75	9f	45
e0	02	e3	5a	bd	ad	41	f7	1d	d7	31	83	63	71	9a	27	ca
f0	01	fd	75	8a	ea	1c	9f	6a	5f	ac	2d	dd	bc	45	cf	35

TABLE B.1 – Propriété de la boîte-S de PICARO : les colonnes sont égales (sur leurs quartets bas) par groupe de 3, de même que les lignes.

Démonstration. Supposons que l'on ait fixé une colonne b de la table de correspondance de S et montrons qu'il y a exactement 3 coefficients dans cette colonne possédant le même quartet bas. Soit $n \in \mathbb{F}_{2^4}$ avec $n = (a^3 + \texttt{0x02})(b^3 + \texttt{0x04})$ un des quartets bas de la colonne. Obtenir un autre coefficient avec le même quartet bas correspond à résoudre en x l'équation :

$$\begin{aligned} & (a^3 + \texttt{0x02})(b^3 + \texttt{0x04}) = (x^3 + \texttt{0x02})(b^3 + \texttt{0x04}) \\ \Leftrightarrow \quad & x^3 - a^3 = 0 \end{aligned} \tag{B.1}$$

(On note que $\forall b \in \mathbb{F}_{2^4}, \quad b^3 + \texttt{0x04} \neq 0$).
Une première racine évidente étant a on réécrit B.1 en :

$$(x - a)(x^2 + ax + a^2) = 0 \tag{B.2}$$

Par changement de variable, en posant $Y = \frac{x}{a}$, on obtient :

$$(B.1) \Leftrightarrow (Y - 1)(Y^2 + Y + 1) = 0 \tag{B.3}$$

Ces deux polynômes sont minimaux sur $\mathbb{F}_{2^4}$ et ont comme racines les éléments d'une classe de conjugaison par l'endomorphisme de Fröbenius. Si on note α une racine primitive, les deux classes sont $\{1\}$ et $\{\alpha^5, \alpha^{10}\}$.

$$(B.1) \Leftrightarrow (Y - 1)(Y - \alpha^5)(Y - \alpha^{10}) = 0 \tag{B.4}$$
$$(B.1) \Leftrightarrow (x - a)(x - a\alpha^5)(x - a\alpha^{10}) = 0 \tag{B.5}$$

Les 3 solutions sont donc $\{a, a\alpha^5, a\alpha^{10}\}$.

En conclusion, si a est non nul, on aura bien 3 coefficients distincts dans la même colonne partageant le même quartet bas. [1]

Comme les valeurs des deux autres coefficients (α^5, α^{10}) ne dépendent pas de la colonne ou de la ligne fixée mais uniquement de la première solution a, cela implique qu'on a les colonnes et les lignes qui sont égales sur les quartets bas par groupe de 3.

□

Propriété B.3. *Chaque coefficient qui ne soit pas l'image de* 00 *apparaît 3 fois dans la table. Comme illustré à la table B.1, ceux-ci se positionnent à l'intersection des 3 colonnes et des 3 lignes possédant les mêmes quartets bas.*

Démonstration. Considérons un élément donné de la table qui soit l'image de (a, b) par S. Ses 3 colonnes associées correspondent à a, $a\alpha^5$ et $a\alpha^{10}$, et ses 3 lignes associées correspondent à b, $b\alpha^5$ et $b\alpha^{10}$. Le calcul des images des coefficients correspondant aux 9 intersections, ne donne que 3 valeurs distinctes, comme l'illustre la table B.2.

1. Sur le même principe, on peut montrer que si on fixe un élément d'une ligne, on trouvera 2 autres coefficients partageant les mêmes quartets bas dans cette ligne.

	..	$a\alpha^{10}$		$a\alpha^{5}$		a	
b		$ab\alpha^{10}\|\|(ab)^3$		$ab\alpha^{5}\|\|(ab)^3$		$ab\ \|\|(ab)^3$	
..							
$b\alpha^{5}$		$ab\ \|\|(ab)^3$		$ab\alpha^{10}\|\|(ab)^3$		$ab\alpha^{5}\|\|(ab)^3$	
		..					
$b\alpha^{10}$		$ab\alpha^{5}\|\|(ab)^3$		$ab\ \|\|(ab)^3$		$ab\alpha^{10}\|\|(ab)^3$	

TABLE B.2 – Explication de la structure de la boîte-S de PICARO : les 9 intersections définissent 3 valeurs différentes répétées 3 fois.

□

Table des figures

Liste des tableaux

Bibliographie

[ABL+09] François Arnault, Thierry P. Berger, Cédric Lauradoux, Marine Minier, and Benjamin Pousse. A New Approach for FCSRs. In Michael J. Jacobson Jr., Vincent Rijmen, and Reihaneh Safavi-Naini, editors, *Selected Areas in Cryptography, 16th Annual International Workshop, SAC 2009, Calgary, Alberta, Canada, August 13-14, 2009, Revised Selected Papers*, volume 5867 of *Lecture Notes in Computer Science*, pages 433–448. Springer, 2009.

[AC09] Martin R. Albrecht and Carlos Cid. Algebraic Techniques in Differential Cryptanalysis. In Orr Dunkelman, editor, *Fast Software Encryption, 16th International Workshop, FSE 2009, Leuven, Belgium, February 22-25, 2009, Revised Selected Papers*, volume 5665 of *Lecture Notes in Computer Science*, pages 193–208. Springer, 2009.

[AFL+12] Farzaneh Abed, Christian Forler, Eik List, Stefan Lucks, and Jakob Wenzel. Biclique Cryptanalysis of PRESENT, LED, and KLEIN. Cryptology ePrint Archive, Report 2012/591, 2012.

[AM15] Frederik Armknecht and Vasily Mikhalev. On Lightweight Stream Ciphers with Shorter Internal States. In Gregor Leander, editor, *Fast Software Encryption - 22nd International Workshop, FSE 2015, Istanbul, Turkey, March 8-11, 2015, Revised Selected Papers*, volume 9054 of *Lecture Notes in Computer Science*, pages 451–470. Springer, 2015.

[ANS11] Jean-Philippe Aumasson, María Naya-Plasencia, and Markku-Juhani O. Saarinen. Practical Attack on 8 Rounds of the Lightweight Block Cipher KLEIN. In Daniel J. Bernstein and Sanjit Chatterjee, editors, *Progress in Cryptology - INDOCRYPT 2011 - 12th International Conference on Cryptology in India, Chennai, India, December 11-14, 2011. Proceedings*, volume 7107 of *Lecture Notes in Computer Science*, pages 134–145. Springer, 2011.

[ARS+15] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In Oswald and Fischlin [OF15], pages 430–454.

[ASA15] Zahra Ahmadian, Mahmoud Salmasizadeh, and Mohammad Reza Aref. Biclique Cryptanalysis of the Full-Round KLEIN Block Cipher. *IET Information Security*, 9(5) :294–301, 2015.

[Bab95] Steve Babbage. Improved "Exhaustive Search" Attacks on Stream Ciphers. In *Security and Detection, 1995., European Convention on*, pages 161–166. IET, 1995.

[Ban15] Subhadeep Banik. Some Results on Sprout. In Alex Biryukov and Vipul Goyal, editors, *Progress in Cryptology - INDOCRYPT 2015 - 16th International Conference on Cryptology in India, Bangalore, India, December 6-9, 2015, Proceedings*, volume 9462 of *Lecture Notes in Computer Science*, pages 124–139. Springer, 2015.

[BBI+15] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita, and Francesco Regazzoni. Midori : A block cipher for low energy. In Iwata and Cheon [IC15], pages 411–436.

[BBS99] Eli Biham, Alex Biryukov, and Adi Shamir. Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials. In Jacques Stern, editor, *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, volume 1592 of *Lecture Notes in Computer Science*, pages 12–23. Springer, 1999.

[BC04] Eli Biham and Rafi Chen. Near-Collisions of SHA-0. In Franklin [Fra04], pages 290–305.

[BCG+12] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. PRINCE - A low-latency block cipher for pervasive computing applications - extended abstract. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, volume 7658 of *Lecture Notes in Computer Science*, pages 208–225. Springer, 2012.

[BCP97] Wieb Bosma, John Cannon, and Catherine Playoust. The Magma Algebra System I : The User Language. *Journal of Symbolic Computation*, 24(3) :235 – 265, 1997.

[BCQ04] Alex Biryukov, Christophe De Cannière, and Michaël Quisquater. On Multiple Linear Approximations. In Franklin [Fra04], pages 1–22.

[BD08a] Steve Babbage and Matthew Dodd. The MICKEY Stream Ciphers. In Robshaw and Billet [RB08], pages 191–209.

[BD08b] Steve Babbage and Matthew Dodd. The MICKEY Stream Ciphers. In Robshaw and Billet [RB08], pages 191–209.

[BDD+15] Achiya Bar-On, Itai Dinur, Orr Dunkelman, Virginie Lallemand, Nathan Keller, and Boaz Tsaban. Cryptanalysis of SP Networks with Partial Non-Linear Layers. In Oswald and Fischlin [OF15], pages 315–342.

[BDF11] Charles Bouillaguet, Patrick Derbez, and Pierre-Alain Fouque. Automatic Search of Attacks on Round-Reduced AES and Applications. In Rogaway [Rog11], pages 169–187.

[BG11] Céline Blondeau and Benoît Gérard. Multiple Differential Cryptanalysis : Theory and Practice. In Antoine Joux, editor, *Fast Software Encryption - 18th International Workshop, FSE 2011, Lyngby, Denmark, February 13-16, 2011,*

Revised Selected Papers, volume 6733 of *Lecture Notes in Computer Science*, pages 35–54. Springer, 2011.

[Bih94] Eli Biham. New Types of Cryptanalytic Attacks Using Related Keys. *Journal of Cryptology*, 7(4) :229–246, 1994.

[BKL+07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT : An Ultra-Lightweight Block Cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, 2007.

[Blo11] Céline Blondeau. *La cryptanalyse différentielle et ses généralisations. (Differential cryptanalysis and its generalizations)*. PhD thesis, Pierre and Marie Curie University, Paris, France, 2011.

[BNS14] Christina Boura, María Naya-Plasencia, and Valentin Suder. Scrutinizing and Improving Impossible Differential Attacks : Applications to CLEFIA, Camellia, LBlock and Simon. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 179–199. Springer, 2014.

[BP15] Alex Biryukov and Léo Perrin. Lightweight Cryptography Lounge. http://cryptolux.org/index.php/Lightweight_Cryptography, 2015.

[BR14] Andrey Bogdanov and Vincent Rijmen. Linear hulls with correlation zero and linear cryptanalysis of block ciphers. *Des. Codes Cryptography*, 70(3) :369–383, 2014.

[BS90] Eli Biham and Adi Shamir. Differential Cryptanalysis of DES-like Cryptosystems. In Alfred Menezes and Scott A. Vanstone, editors, *Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings*, volume 537 of *Lecture Notes in Computer Science*, pages 2–21. Springer, 1990.

[BS15] Adnan Baysal and Suhap Sahin. RoadRunneR : A Small And Fast Bitslice Block Cipher For Low Cost 8-bit Processors. *IACR Cryptology ePrint Archive*, 2015 :906, 2015.

[BSS+15] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK lightweight block ciphers. In *Proceedings of the 52nd Annual Design Automation Conference, San Francisco, CA, USA, June 7-11, 2015*, pages 175 :1–175 :6. ACM, 2015.

[BW99] Alex Biryukov and David Wagner. Slide Attacks. In *Fast Software Encryption - FSE'99*, volume 1636 of *Lecture Notes in Computer Science*, pages 245–259. Springer, 1999.

[Can06] Christophe De Cannière. Trivium : A Stream Cipher Construction Inspired by Block Cipher Design Principles. In Sokratis K. Katsikas, Javier Lopez, Michael Backes, Stefanos Gritzalis, and Bart Preneel, editors, *Information Security, 9th*

International Conference, ISC 2006, Samos Island, Greece, August 30 - September 2, 2006, Proceedings, volume 4176 of *Lecture Notes in Computer Science*, pages 171–186. Springer, 2006.

[Car10] Claude Carlet. Boolean Functions for Cryptography and Error Correcting Codes. *Boolean Models and Methods in Mathematics, Computer Science, and Engineering*, 2 :257–397, 2010.

[Car11] Claude Carlet. Relating Three Nonlinearity Parameters of Vectorial Functions and Building APN functions from Bent Functions. *Designs, Codes and Cryptography*, 59(1-3) :89–109, 2011.

[CC98] Anne Canteaut and Florent Chabaud. A New Algorithm for Finding Minimum-Weight Words in a Linear Code : Application to McEliece's Cryptosystem and to Narrow-Sense BCH Codes of Length 511. *IEEE Transactions on Information Theory*, 44(1) :367–378, 1998.

[CCF+16] Anne Canteaut, Sergiu Carpov, Caroline Fontaine, Tancrède Lepoint, María Naya-Plasencia, Pascal Paillier, and Renaud Sirdey. Stream ciphers : A Practical Solution for Efficient Homomorphic-Ciphertext Compression. In Thomas Peyrin, editor, *Fast Software Encryption - 23st International Workshop, FSE 2016, Bochum, Germany, March 20-23, 2016. Revised Selected Papers*, volume 9783 of *Lecture Notes in Computer Science*. Springer, 2016.

[CDK09] Christophe De Cannière, Orr Dunkelman, and Miroslav Knezevic. KATAN and KTANTAN - A Family of Small and Efficient Hardware-Oriented Block Ciphers. In Christophe Clavier and Kris Gaj, editors, *Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings*, volume 5747 of *Lecture Notes in Computer Science*, pages 272–288. Springer, 2009.

[CHN08] Joo Yeon Cho, Miia Hermelin, and Kaisa Nyberg. A New Technique for Multidimensional Linear Cryptanalysis with Applications on Reduced Round Serpent. In Pil Joong Lee and Jung Hee Cheon, editors, *Information Security and Cryptology - ICISC 2008, 11th International Conference, Seoul, Korea, December 3-5, 2008, Revised Selected Papers*, volume 5461 of *Lecture Notes in Computer Science*, pages 383–398. Springer, 2008.

[CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.

[CKPS00] Nicolas Courtois, Alexander Klimov, Jacques Patarin, and Adi Shamir. Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations. In Bart Preneel, editor, *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, volume 1807 of *Lecture Notes in Computer Science*, pages 392–407. Springer, 2000.

[CLN15] Anne Canteaut, Virginie Lallemand, and María Naya-Plasencia. Related-Key Attack on Full-Round PICARO. In Orr Dunkelman and Liam Keliher, editors,

Selected Areas in Cryptography - SAC 2015 - 22nd International Conference, Sackville, NB, Canada, August 12-14, 2015, Revised Selected Papers, volume 9566 of *Lecture Notes in Computer Science*, pages 86–101. Springer, 2015.

[CM03] Nicolas Courtois and Willi Meier. Algebraic Attacks on Stream Ciphers with Linear Feedback. In Eli Biham, editor, *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*, volume 2656 of *Lecture Notes in Computer Science*, pages 345–359. Springer, 2003.

[CP02] Nicolas Courtois and Josef Pieprzyk. Cryptanalysis of Block Ciphers with Overdefined Systems of Equations. In Yuliang Zheng, editor, *Advances in Cryptology - ASIACRYPT 2002, 8th International Conference on the Theory and Application of Cryptology and Information Security, Queenstown, New Zealand, December 1-5, 2002, Proceedings*, volume 2501 of *Lecture Notes in Computer Science*, pages 267–287. Springer, 2002.

[DEM15] Christoph Dobraunig, Maria Eichlseder, and Florian Mendel. Higher-order cryptanalysis of LowMC. *IACR Cryptology ePrint Archive*, 2015 :407, 2015.

[DH76] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6) :644–654, 1976.

[DH77] Whitfield Diffie and Martin E. Hellman. Special Feature Exhaustive Cryptanalysis of the NBS Data Encryption Standard. *IEEE Computer*, 10(6) :74–84, 1977.

[DLMW15] Itai Dinur, Yunwen Liu, Willi Meier, and Qingju Wang. Optimized Interpolation Attacks on LowMC. In Iwata and Cheon [IC15], pages 535–560.

[DLR16] Sébastien Duval, Virginie Lallemand, and Yann Rotella. Cryptanalysis of the FLIP Family of Stream Ciphers. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016 - 36th Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I*, volume 9814 of *Lecture Notes in Computer Science*. Springer, 2016.

[EJ00] Patrik Ekdahl and Thomas Johansson. SNOW-a New Stream Cipher. In *Proceedings of First Open NESSIE Workshop, KU-Leuven*, pages 167–168, 2000.

[EK15] Muhammed F. Esgin and Orhun Kara. Practical Cryptanalysis of Full Sprout with TMD Tradeoff Attacks. *IACR Cryptology ePrint Archive*, 2015 :289, 2015.

[Fau99] Jean-Charles Faugere. A new efficient algorithm for computing Gröbner bases (F 4). *Journal of pure and applied algebra*, 139(1) :61–88, 1999.

[Fei74] Horst Feistel. Block Cipher Cryptographic System, 1974. U.S. Patent 3.798.359.

[FIP77] FIPS. Data Encryption Standard. *Federal Information Processing Standards, National Bureau of Standards, US Department of Commerce*, 1977.

[Fra04] Matthew K. Franklin, editor. *Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, volume 3152 of *Lecture Notes in Computer Science*. Springer, 2004.

[GB08] Tim Good and Mohammed Benaissa. Hardware Performance of eSTREAM Phase-III Stream Cipher Candidates. In *State of the Art of Stream Ciphers Workshop (SASC 2008)*, pages 163–173, 2008.

[Gef73] Philip R Geffe. How to protect data with ciphers that are really hard to break. *Electronics*, 46(1) :99–101, 1973.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 169–178. ACM, 2009.

[GGNS13] Benoît Gérard, Vincent Grosso, María Naya-Plasencia, and François-Xavier Standaert. Block Ciphers That Are Easier to Mask : How Far Can We Go ? In Guido Bertoni and Jean-Sébastien Coron, editors, *Cryptographic Hardware and Embedded Systems - CHES 2013 - 15th International Workshop, Santa Barbara, CA, USA, August 20-23, 2013. Proceedings*, volume 8086 of *Lecture Notes in Computer Science*, pages 383–399. Springer, 2013.

[GNL11] Zheng Gong, Svetla Nikova, and Yee Wei Law. KLEIN : A New Family of Lightweight Block Ciphers. In Ari Juels and Christof Paar, editors, *RFID. Security and Privacy - 7th International Workshop, RFIDSec 2011, Amherst, USA, June 26-28, 2011, Revised Selected Papers*, volume 7055 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2011.

[GNPW13] Jian Guo, Ivica Nikolic, Thomas Peyrin, and Lei Wang. Cryptanalysis of Zorro. Cryptology ePrint Archive, Report 2013/713, 2013.

[Gol97] Jovan Dj. Golic. Cryptanalysis of Alleged A5 Stream Cipher. In Walter Fumy, editor, *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, volume 1233 of *Lecture Notes in Computer Science*, pages 239–255. Springer, 1997.

[GP99] Louis Goubin and Jacques Patarin. DES and differential power analysis (the "duplication" method). In Çetin Kaya Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99, Worcester, MA, USA, August 12-13, 1999, Proceedings*, volume 1717 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 1999.

[GPPR11] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The LED Block Cipher. In Preneel and Takagi [PT11], pages 326–341.

[GST14] Daniel Genkin, Adi Shamir, and Eran Tromer. RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 444–461. Springer, 2014.

[Hao15] Yonglin Hao. A Related-Key Chosen-IV Distinguishing Attack on Full Sprout Stream Cipher. *IACR Cryptology ePrint Archive*, 2015 :231, 2015.

[Hel80] Martin E. Hellman. A Cryptanalytic Time-Memory Trade-off. *IEEE Transactions on Information Theory*, 26(4) :401–406, 1980.

[HJM07] Martin Hell, Thomas Johansson, and Willi Meier. Grain : a Stream Cipher for Constrained Environments. *IJWMC*, 2(1) :86–93, 2007.

[IC15] Tetsu Iwata and Jung Hee Cheon, editors. *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II*, volume 9453 of *Lecture Notes in Computer Science*. Springer, 2015.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private Circuits : Securing Hardware against Probing Attacks. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.

[Ker83] Auguste Kerckhoffs. La cryptographie militaire. *Journal des sciences militaires*, IX :5–38, 161–191, 1883.

[KLPR10] Lars R. Knudsen, Gregor Leander, Axel Poschmann, and Matthew J. B. Robshaw. PRINTcipher : A Block Cipher for IC-Printing. In Mangard and Standaert [MS10], pages 16–32.

[KMP+98] Lars R. Knudsen, Willi Meier, Bart Preneel, Vincent Rijmen, and Sven Verdoolaege. Analysis Methods for (Alleged) RC4. In Kazuo Ohta and Dingyi Pei, editors, *Advances in Cryptology - ASIACRYPT '98, International Conference on the Theory and Applications of Cryptology and Information Security, Beijing, China, October 18-22, 1998, Proceedings*, volume 1514 of *Lecture Notes in Computer Science*, pages 327–341. Springer, 1998.

[Knu69] Donald E. Knuth. *The Art of Computer Programming, Volume II : Seminumerical Algorithms*. Addison-Wesley, 1969.

[Knu94] Lars R. Knudsen. Truncated and Higher Order Differentials. In Bart Preneel, editor, *Fast Software Encryption : Second International Workshop. Leuven, Belgium, 14-16 December 1994, Proceedings*, volume 1008 of *Lecture Notes in Computer Science*, pages 196–211. Springer, 1994.

[Knu98] Lars R. Knudsen. DEAL - A 128-bit Block Cipher, AES submission, 1998. Department of Informatics, University of Bergen, Norway.

[Koc96] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

[KR07] Lars R. Knudsen and Vincent Rijmen. Known-Key Distinguishers for Some Block Ciphers. In Kaoru Kurosawa, editor, *Advances in Cryptology - ASIACRYPT 2007, 13th International Conference on the Theory and Application of Cryptology and Information Security, Kuching, Malaysia, December 2-6, 2007, Proceedings*, volume 4833 of *Lecture Notes in Computer Science*, pages 315–324. Springer, 2007.

[Lai94] Xuejia Lai. *Higher Order Derivatives and Differential Cryptanalysis*, pages 227–233. Springer US, Boston, MA, 1994.

[LH94] Susan K. Langford and Martin E. Hellman. Differential-Linear Cryptanalysis. In Yvo Desmedt, editor, *Advances in Cryptology - CRYPTO '94, 14th Annual*

International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings, volume 839 of *Lecture Notes in Computer Science*, pages 17–25. Springer, 1994.

[LMM91] Xuejia Lai, James L. Massey, and Sean Murphy. Markov ciphers and differential cryptanalysis. In Donald W. Davies, editor, *Advances in Cryptology - EUROCRYPT '91, Workshop on the Theory and Application of of Cryptographic Techniques, Brighton, UK, April 8-11, 1991, Proceedings*, volume 547 of *Lecture Notes in Computer Science*, pages 17–38. Springer, 1991.

[LN83] Rudolf Lidl and Harald Niederreiter. *Finite Fields*. Cambridge university press, 1983.

[LN14] Virginie Lallemand and María Naya-Plasencia. Cryptanalysis of KLEIN. In Carlos Cid and Christian Rechberger, editors, *Fast Software Encryption - 21st International Workshop, FSE 2014, London, UK, March 3-5, 2014. Revised Selected Papers*, volume 8540 of *Lecture Notes in Computer Science*, pages 451–470. Springer, 2014.

[LN15] Virginie Lallemand and María Naya-Plasencia. Cryptanalysis of Full Sprout. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 663–682. Springer, 2015.

[LT16] Petr Lisonek and Layla Trummer. Algorithms for the minimum weight of linear codes. *Adv. in Math. of Comm.*, 10(1) :195–207, 2016.

[MAM16] Vasily Mikhalev, Frederik Armknecht, and Christian Müller. On Ciphers that Continuously Access the Non-Volatile Key, 2016. http://materials.dagstuhl.de/files/16/16021/16021.FrederikArmknecht.Slides.pdf.

[Mas69] James L. Massey. Shift-register synthesis and BCH decoding. *IEEE Transactions on Information Theory*, 15(1) :122–127, 1969.

[Mat93] Mitsuru Matsui. Linear Cryptanalysis Method for DES Cipher. In Tor Helleseth, editor, *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings*, volume 765 of *Lecture Notes in Computer Science*, pages 386–397. Springer, 1993.

[MJSC15] Pierrick Méaux, Anthony Journault, François-Xavier Standaert, and Claude Carlet. Towards Stream Ciphers for Efficient FHE with Low-Noise Ciphertexts. *personal communication*, October 2015.

[MJSC16] Pierrick Méaux, Anthony Journault, François-Xavier Standaert, and Claude Carlet. Towards Stream Ciphers for Efficient FHE with Low-Noise Ciphertexts. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I*, volume 9665 of *Lecture Notes in Computer Science*, pages 311–343. Springer, 2016.

[MS88] Willi Meier and Othmar Staffelbach. Fast Correlation Attacks on Stream Ciphers (Extended Abstract). In Christoph G. Günther, editor, *Advances in Cryptology -*

EUROCRYPT '88, Workshop on the Theory and Application of of Cryptographic Techniques, Davos, Switzerland, May 25-27, 1988, Proceedings, volume 330 of *Lecture Notes in Computer Science*, pages 301–314. Springer, 1988.

[MS10] Stefan Mangard and François-Xavier Standaert, editors. *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, volume 6225 of *Lecture Notes in Computer Science*. Springer, 2010.

[MSBD15] Subhamoy Maitra, Santanu Sarkar, Anubhab Baksi, and Pramit Dey. Key Recovery from State Information of Sprout : Application to Cryptanalysis and Fault Attack. *IACR Cryptology ePrint Archive*, 2015 :236, 2015.

[MvOV96] Alfred Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[MWGP11] Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. Differential and Linear Cryptanalysis Using Mixed-Integer Linear Programming. In Wu et al. [WYL12], pages 57–76.

[Nay11] María Naya-Plasencia. How to Improve Rebound Attacks. In Rogaway [Rog11], pages 188–205.

[NLV11] Michael Naehrig, Kristin E. Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical ? In Christian Cachin and Thomas Ristenpart, editors, *Proceedings of the 3rd ACM Cloud Computing Security Workshop, CCSW 2011, Chicago, IL, USA, October 21, 2011*, pages 113–124. ACM, 2011.

[NWW15] Ivica Nikolic, Lei Wang, and Shuang Wu. The Parallel-Cut Meet-in-the-Middle Attack. *Cryptography and Communications*, 7(3) :331–345, 2015.

[Nyb96] Kaisa Nyberg. Generalized Feistel Networks. In Kwangjo Kim and Tsutomu Matsumoto, editors, *Advances in Cryptology - ASIACRYPT '96, International Conference on the Theory and Applications of Cryptology and Information Security, Kyongju, Korea, November 3-7, 1996, Proceedings*, volume 1163 of *Lecture Notes in Computer Science*, pages 91–104. Springer, 1996.

[OF15] Elisabeth Oswald and Marc Fischlin, editors. *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*. Springer, 2015.

[PRC12a] Gilles Piret, Thomas Roche, and Claude Carlet. PICARO - A Block Cipher Allowing Efficient Higher-Order Side-Channel Resistance. In *Applied Cryptography and Network Security - ACNS 2012*, volume 7341 of *Lecture Notes in Computer Science*, pages 311–328. Springer, 2012.

[PRC12b] Gilles Piret, Thomas Roche, and Claude Carlet. PICARO - A Block Cipher Allowing Efficient Higher-Order Side-Channel Resistance - Extended Version -. *IACR Cryptology ePrint Archive*, 2012 :358, 2012.

[PT11] Bart Preneel and Tsuyoshi Takagi, editors. *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, volume 6917 of *Lecture Notes in Computer Science*. Springer, 2011.

[RASA14] Shahram Rasoolzadeh, Zahra Ahmadian, Mahmoud Salmasizadeh, and Mohammad Reza Aref. Total Break of Zorro using Linear and Differential Attacks. *IACR Cryptology ePrint Archive*, 2014 :220, 2014.

[RB08] Matthew J. B. Robshaw and Olivier Billet, editors. *New Stream Cipher Designs - The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*. Springer, 2008.

[RM16] Dibyendu Roy and Sourav Mukhopadhyay. Fault analysis and weak key-IV attack on Sprout. *IACR Cryptology ePrint Archive*, 2016 :207, 2016.

[Rog11] Phillip Rogaway, editor. *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*. Springer, 2011.

[RP10] Matthieu Rivain and Emmanuel Prouff. Provably Secure Higher-Order Masking of AES. In Mangard and Standaert [MS10], pages 413–427.

[RPLP08] Carsten Rolfes, Axel Poschmann, Gregor Leander, and Christof Paar. Ultra-Lightweight Implementations for Smart Devices - Security for 1000 Gate Equivalents. In Gilles Grimaud and François-Xavier Standaert, editors, *Smart Card Research and Advanced Applications, 8th IFIP WG 8.8/11.2 International Conference, CARDIS 2008, London, UK, September 8-11, 2008. Proceedings*, volume 5189 of *Lecture Notes in Computer Science*, pages 89–103. Springer, 2008.

[Sha49] Claude E. Shannon. Communication theory of secrecy systems. *Bell system technical journal*, 28(4) :656–715, 1949.

[Sha79] Adi Shamir. How to Share a Secret. *Commun. ACM*, 22(11) :612–613, 1979.

[Sie85] Thomas Siegenthaler. Decrypting a Class of Stream Ciphers Using Ciphertext Only. *IEEE Trans. Computers*, 34(1) :81–85, 1985.

[SIH+11] Kyoji Shibutani, Takanori Isobe, Harunaga Hiwatari, Atsushi Mitsuda, Toru Akishita, and Taizo Shirai. Piccolo : An Ultra-Lightweight Blockcipher. In Preneel and Takagi [PT11], pages 342–357.

[SK96] Bruce Schneier and John Kelsey. Unbalanced Feistel Networks and Block Cipher Design. In Dieter Gollmann, editor, *Fast Software Encryption, Third International Workshop, Cambridge, UK, February 21-23, 1996, Proceedings*, volume 1039 of *Lecture Notes in Computer Science*, pages 121–144. Springer, 1996.

[SMMK12] Tomoyasu Suzaki, Kazuhiko Minematsu, Sumio Morioka, and Eita Kobayashi. TWINE : A Lightweight Block Cipher for Multiple Platforms. In Lars R. Knudsen and Huapeng Wu, editors, *Selected Areas in Cryptography, 19th International Conference, SAC 2012, Windsor, ON, Canada, August 15-16, 2012, Revised Selected Papers*, volume 7707 of *Lecture Notes in Computer Science*, pages 339–354. Springer, 2012.

[SPQ03] Francois-Xavier Standaert, Gilles Piret, and Jean-Jacques Quisquater. Cryptanalysis of Block Ciphers : A Survey. *UCL Crypto Group*, 2003.

[SSA+07] Taizo Shirai, Kyoji Shibutani, Toru Akishita, Shiho Moriai, and Tetsu Iwata. The 128-Bit Blockcipher CLEFIA (Extended Abstract). In Alex Biryukov, editor, *Fast Software Encryption, 14th International Workshop, FSE 2007, Luxembourg, Luxembourg, March 26-28, 2007, Revised Selected Papers*, volume 4593 of *Lecture Notes in Computer Science*, pages 181–195. Springer, 2007.

[TG91] Anne Tardy-Corfdir and Henri Gilbert. A Known Plaintext Attack of FEAL-4 and FEAL-6. In Joan Feigenbaum, editor, *Advances in Cryptology - CRYPTO'91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 172–181. Springer, 1991.

[Vé15] Pascal Véron. Journées Codage et Cryptographie 2015 - JC2 2015 - 12ème édition des Journées Codage et Cryptographie du GT C2, 5 octobre au 9 octobre 2015, La Londe-les-Maures, France. 2015.

[Wag99] David Wagner. The Boomerang Attack. In Lars R. Knudsen, editor, *Fast Software Encryption, 6th International Workshop, FSE '99, Rome, Italy, March 24-26, 1999, Proceedings*, volume 1636 of *Lecture Notes in Computer Science*, pages 156–170. Springer, 1999.

[WWGY14] Yanfeng Wang, Wenling Wu, Zhiyuan Guo, and Xiaoli Yu. Differential Cryptanalysis and Linear Distinguisher of Full-Round Zorro. In Ioana Boureanu, Philippe Owesarski, and Serge Vaudenay, editors, *Applied Cryptography and Network Security - 12th International Conference, ACNS 2014, Lausanne, Switzerland, June 10-13, 2014. Proceedings*, volume 8479 of *Lecture Notes in Computer Science*, pages 308–323. Springer, 2014.

[WYL12] Chuankun Wu, Moti Yung, and Dongdai Lin, editors. *Information Security and Cryptology - 7th International Conference, Inscrypt 2011, Beijing, China, November 30 - December 3, 2011. Revised Selected Papers*, volume 7537 of *Lecture Notes in Computer Science*. Springer, 2012.

[WZ11] Wenling Wu and Lei Zhang. LBlock : A Lightweight Block Cipher. In Javier Lopez and Gene Tsudik, editors, *Applied Cryptography and Network Security - 9th International Conference, ACNS 2011, Nerja, Spain, June 7-10, 2011. Proceedings*, volume 6715 of *Lecture Notes in Computer Science*, pages 327–344, 2011.

[YWLZ11] Xiaoli Yu, Wenling Wu, Yanjun Li, and Lei Zhang. Cryptanalysis of Reduced-Round KLEIN Block Cipher. In Wu et al. [WYL12], pages 237–250.

[ZG15] Bin Zhang and Xinxin Gong. Another Tradeoff Attack on Sprout-Like Stream Ciphers. In Iwata and Cheon [IC15], pages 561–585.